Lecture Notes:
Computer Networks and Security (2IC60)

T. Ozcelebi and J.I. den Hartog

version 0.38 (2019)

Important Notice:

Students can take their own notes, for example, on lecture slide set PDF documents (available on the course website before each lecture). This document and references in it marked as *required reading* (provided at the end of each chapter) form a supplement to the lectures. They are meant to give more detail and fill the gaps. In the exam, you are responsible for the *lecture content*, the *lecture notes* (this document) and the *required reading*. Lecture slides will often only contain illustrations of main ideas. Further explanation is provided during the lecture, in the *lecture notes* and in *required reading*.

Each chapter of this document provides a literature section that describes *required reading* and *suggested reading*. The required reading is part of the exam material while the suggested reading is not. The suggested reading is for students who are interested in background information and/or a different perspective on or presentation of the material that is useful to help get a better/deeper understanding of the material. Additional references to related work may be given inside the text. These provide related materials and/or more in-depth discussions. Interested students can learn more reading these; but they are considered to be outside the scope of assessment.

Mandatory reading checklist (= part of the exam)

- The lecture slides.
- The lecture notes (this document).
- The *required reading* listed at the end of each chapter.

Optional reading checklist (= not part of the exam)

- The *suggested reading* listed at the end of each chapter.
- Additional references inside the main text.

This is the first iteration of these notes; some flaws will be present and improvement suggestions are welcome. Please report textual mistakes that you come across to the e-mail address `t.ozcelebi@tue.nl` and help us improve the quality.

End of notice.

About this document

This document is meant as supporting material for the lectures in the Computer Networks and Security (2IC60) course and the referenced reading materials. It is not meant to be a stand alone document. This document is still work-in-progress.

Thanks

The course material covered in these notes has been developed with contributions from Elisa Costante, Ricardo Corin, Jeroen Doumen, Sandro Etalle, Pieter Hartel, Boris Skoric, Nicola Zannone, Igor Radovanovic and Johan Lukkien.

The authors would also like to acknowledge the sources of some of the figures and exercises in this document:

- Jim Kurose, Keith Ross [20].
- Behrouz Forouzan [17].

Contents

Chapter 1

Introduction

Document structure

- **Chapter 1** provides a general introduction to the concepts of computer networks and (network) security.
- **Chapter 2** explains how protocols govern computer networks and gives an overview of protocol layering and protocol stacks.
- **Chapter 3** introduces application layer concepts and the principles of networked applications.
- **Chapter 4** introduces transport layer services and widely used transport layer protocols of the Internet.
- **Chapter 5** gives an overview of network layer services for end device to end device data delivery.
- **Chapter 6** introduces the data link layer whose task is to transfer data over a single (wired or wireless) link.
- **Chapter 7** discusses key security issues at the network edge; authorization (what resources can be used at the server side) and authentication (who is present on the client side).
- **Chapter 8** focuses more on the security aspects regarding network core by looking at threats and corresponding countermeasures at the different network layers.
- **Chapter 9** discusses in detail an essential tool for network security: cryptography.
- **Chapter 10** explains how we can analyze the security properties of protocols.

1.1 Networks and Computer Networks

1.1.1 Networks

Before we start our discussion on computer networks, let us take a step back and discuss what a network in general is. The concept of a network is no stranger to

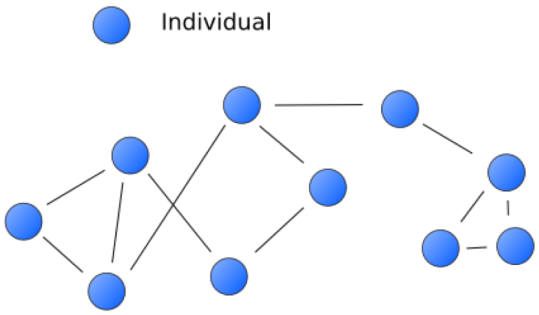


Figure 1.1: An interconnected configuration of (system) components.

the common person. It typically refers to an "interconnected" configuration of individuals. Similarly, the term networking refers to communicating either with or within a group. Among humans, wherever there is communication, there is a network. Examples are many, e.g. a network of neighbors (neighborhood watch), a network of intelligence (spy networks), a network of gossip among close friends, a news network, a broadcast network, a book club or giant social media platforms like **Facebook**, **Twitter** and **Instagram**. The history of networks goes way back; even to the very early periods of the civilization.

The concept of a network is not limited to the human context and it is well-known to the animal kingdom, to plants and even to bacteria. A good example, which inspires many modern networks of wireless embedded devices by the way, is a colony of ants searching for food. Each worker ant has a very simple behavior, which by itself (without communication and collaboration between ants) cannot accomplish much. A worker ant's behavior is leaving a trail of pheromone wherever it goes and following this trail back in the opposite direction once it finds food, carrying some of the food back to the nest, making the trail of pheromone (almost) twice as strong. Without forming a network, this is very inefficient as each ant needs to find food separately and there is a non-negligible risk of getting lost. What ants do instead is that they stop their random walk and join existing trails of pheromone that are relatively stronger as they come across these, adding their pheromone to make it even stronger. As a result, a lot of ants make up crowded *highways* of food traffic. Successful implementation of such emergent behavior by means of this simple protocol is a matter of life and death for ants.

> *"An important observation so far is that very simple network protocols can result in a collective behavior of impressive complexity."*

More generally, a network is an interconnected set of components as shown in Figure 1.1.

1.1.2 Computer Networks

Computer networks play a key role in modern society. We define a computer network from two perspectives: physically (hardware) and logically (software and data).

Physically From a physical (hardware) perspective, a computer network is a hardware infrastructure interconnecting end-devices, where end-devices can be in one of the many possible forms such as personal computers, personal digital assistants, smart phones, wireless sensors, wireless actuators (e.g. a *Philips Hue lamp*) and smart televisions. **End-devices** are typically made up of (embedded) hardware with a Network Interface Card (NIC) and operating systems or basic middleware.

Logically From a software and data perspective, a computer network is a system facilitating information exchange between applications that do not share a physical memory component or memory space.

1.2 Push Behind Networks

We need networks and their services to improve our lives by serving various applications. There are various motivations for the push behind networks. These motivations take their roots at technological developments, industrial developments, economics and other social aspects.

1.2.1 Technological Development

In the 20th century, there were two important (but not very accurate) predictions about the future market of computers. The first one was by the famous Thomas Watson, the president of the IBM (International Business Machines) corporation. In 1943, Thomas Watson said

> *"I think there is a world market for maybe five computers."*
>
> T. Watson

This view was out there for a long time and 34 years later in 1977 another important figure Ken Olsen, the president of DEC (Digital Equipment Corporation), said

> *"There is no reason for any individual to have a computer in their home."*
>
> K. Olsen

Gordon E. Moore, the co-founder of Intel, made a more accurate prediction of what would happen in the future of computer technology. In his article dated 1965[1], he made the observation that the number of circuit components on an integrated circuit doubles every year and he projected that this would continue to be the case also in the future. Years later, he made a revision to this projection. In practice, what is widely known as Moore's law (*Moore's observation* may be a better term by the way) translates to the following and is depicted in Figure 1.2.

> ***Moore's Law:*** *"The number of transistors that fit in unit area of an integrated circuit (or, equivalently, the total processing power of computers) doubles every two years."*

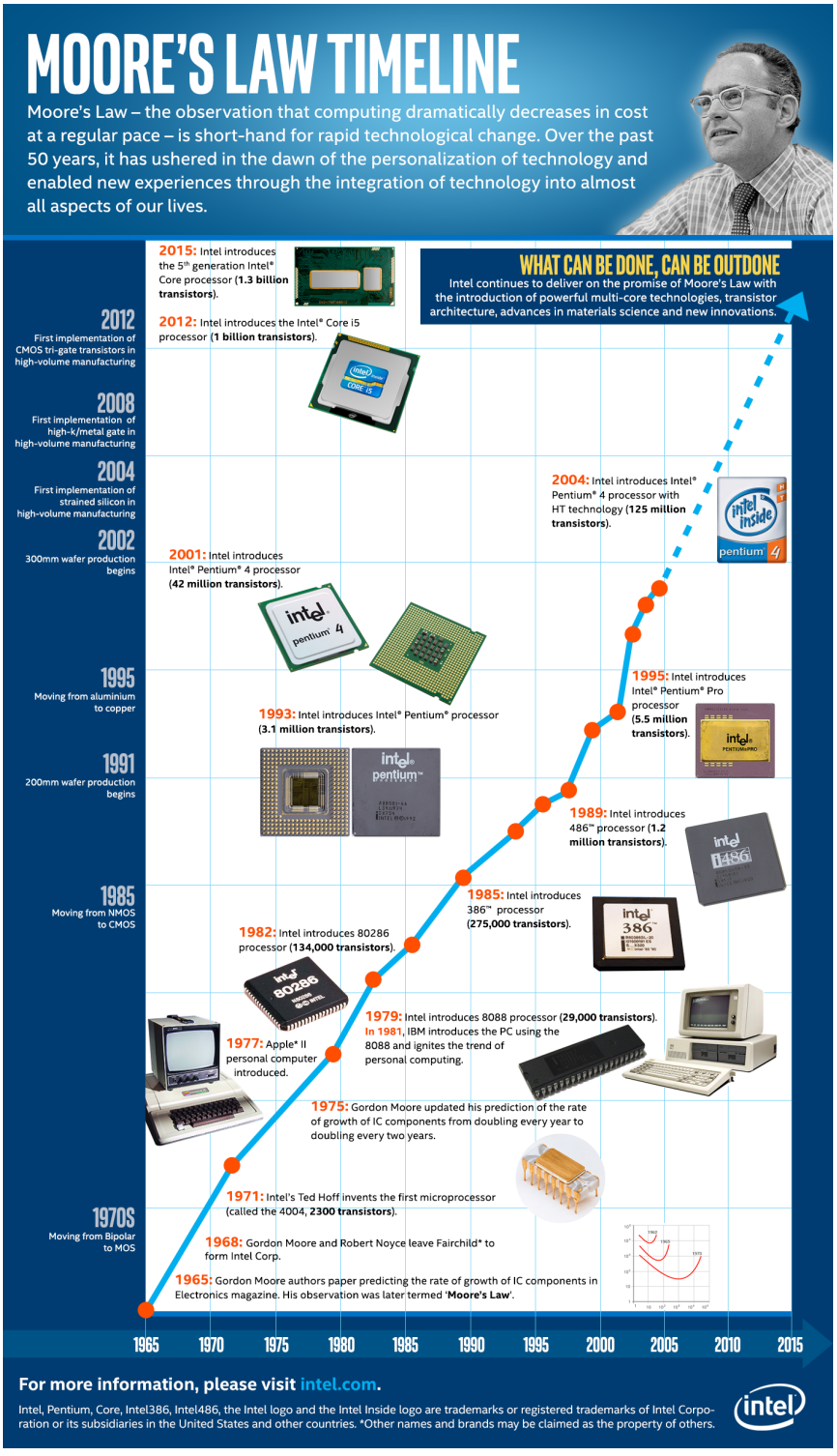


Figure 1.2: To this day, the number of transistors that fit in an integrated circuit follows Moore's law closely. (Figure source: intel.com)

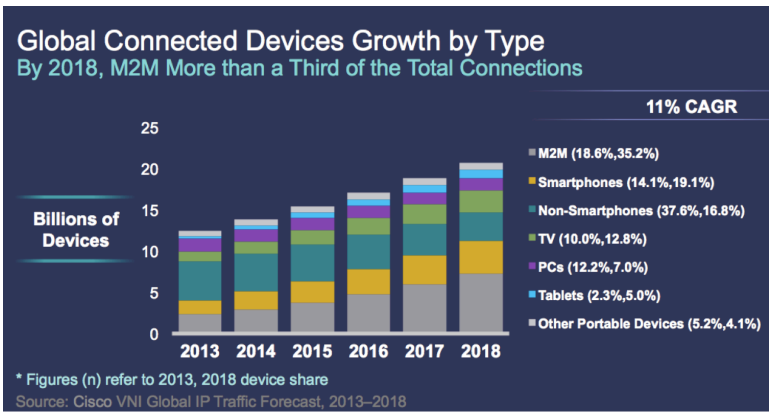


Figure 1.3: The number of devices connected to the Internet and their types across years, with a projection until 2018. (Figure source: Cisco)

Moore's law has an important result. It implies that devices that are just not capable enough to connect to the Internet due to lack of computational power will be powerful enough soon. A recent report by Cisco indicates that we are rapidly moving from the Internet of personal computers to an Internet of smart phones, tablets and machine-to-machine (M2M) communications. All projections are towards a future of the Internet that is dominated by data traffic that does not involve any humans. Figure 1.3 by Cisco shows the profiles of the devices connected to the Internet with projection until 2018.

1.2.2 Industrial Development

The Internet has started with only 4 hosts in 1969, and only in 1983 the number of Internet hosts increased to 500. Today, there are more machines connected to the Internet than there are humans on the planet. Internet usage is rapidly spreading to all classes of the society, in most countries, especially in the developed countries. According to the International Telecommunications Union (ITU), the percentage of Internet users among inhabitants in different parts of the world is as given in Figure 1.4. The increasing reach to the end user also means an increase of the Internet technology demand for the industry.

Metcalfe's law, named after the Ethernet co-inventor Robert Metcalfe, gives an explanation for the industry boom in and around the Internet technology.

> ***Metcalfe's Law:*** *"The value, usefulness, or utility of a network equals the square of the number of users (or connected devices)."*

[1] Gordon E. Moore, "Cramming more components onto integrated circuits", April 1965.

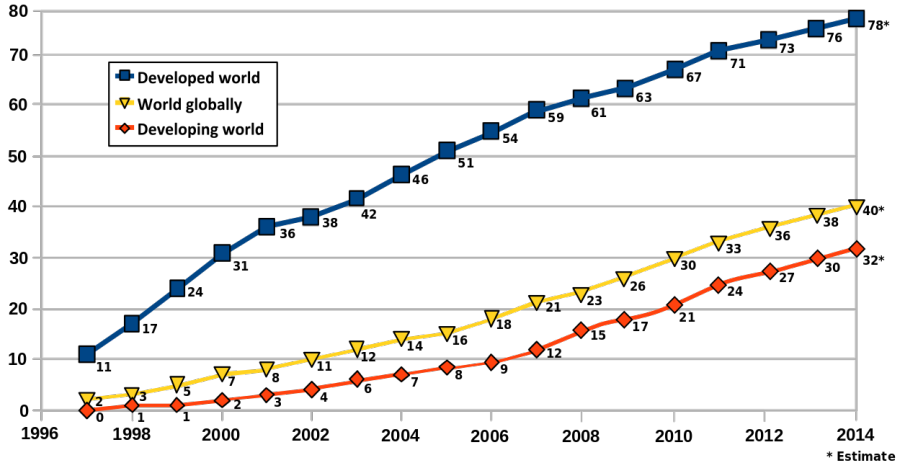


Figure 1.4: The total reach of the Internet technology is increasing rapidly, in parallel with the industry that is associated with it. (Figure source: ITU)

Tag	Full name	Example
B2C	Business-to-consumer	Ordering books on-line
B2B	Business-to-business	Car manufacturer ordering tires from supplier
G2C	Government-to-consumer	Government distributing tax forms electronically
C2C	Consumer-to-consumer	Auctioning second-hand products on line
P2P	Peer-to-peer	File sharing

Figure 1.5: Some e-business types and their examples.

1.2.3 Economic and Social Aspects

In parallel with the rapid rise in the Internet usage all around the world, the e-commerce volume is increasing at a staggering rate. E-businesses can be of many forms, as exemplified in Figure 1.5.

Figure 1.6 shows the worldwide B2C e-commerce sales volumes with a projection of 2.4 trillion US dollars in 2018. According to e-commerce news[2], the Dutch B2C e-commerce has reached a total volume of 16.5 billion euros in 2015 (a realistic projection). According to the same article, there were 3.3 million mobile shoppers in the Netherlands in 2015 (an increase of 13 percent in comparison to the previous year).

Societal demands play an important role in the development of the Internet and Internet-based services. Services like e-learning (Coursera[3] has reached 15 million students), better and cheaper communication (e.g. Whatsapp, Skype), working from home, second life and entertainment (e.g. gaming, Netflix) are a few examples. Trends and hypes play an important role. Trying to push new technology for which the need is not clear or for which there is no real demand

[2]http://ecommercenews.eu/ecommerce-in-the-netherlands-expected-to-reach-e16-5bn-in-2015/

[3]Coursera is an Internet platform for massive open online courses. It hosts over a thousand courses from many universities and serves students all around the world.

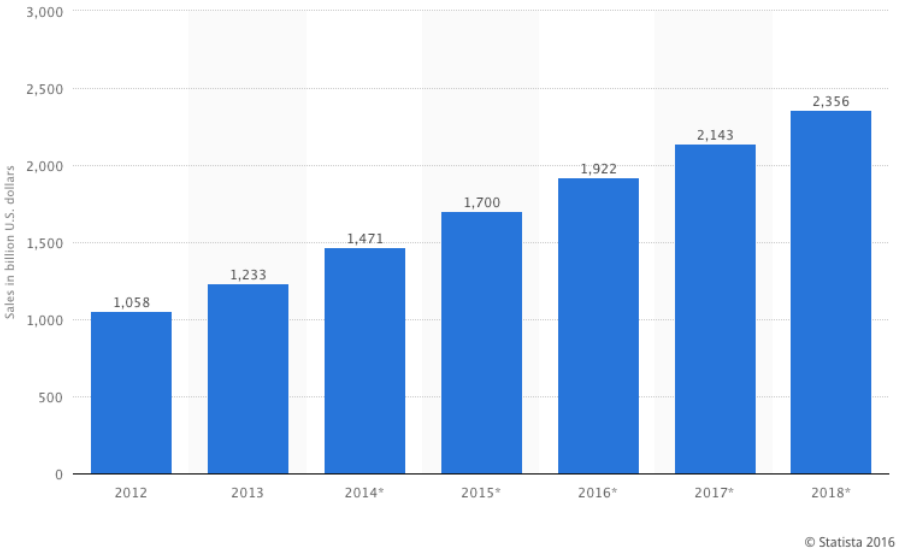


Figure 1.6: Worldwide B2C e-commerce sales volumes. The years marked with a * are projections based on previous data. (Figure source: statista.com)

often does not result in user acceptance. For example, Wireless Application Protocol (WAP) was advertised with the words "Internet made mobile", when in fact it was just a new protocol that did not really have a big impact either socially or economically. The Multimedia Messaging Service (MMS) was advertised as a better replacement for Short Message Service (SMS), but it never could really replace or even come close to SMS. Multimedia integration into text messages became huge only after services like Whatsapp, for which the convenience factor is substantial.

1.3 Standards and Regulations on Networks

Metcalfe's law indicates that the value of a network depends on the size of community it can reach. The Internet can reach a global community thanks to standardization. Standardization (of hardware and protocols) is crucial for device interoperability between different vendors. In this way, there are more suppliers of devices, leading to more competition between suppliers, which results in lower prices for the end user. There are three major standardization bodies for the Internet:

- International Telecommunication Union (ITU)
- Internet Engineering Task Force (IETF)
- Institute of Electrical and Electronic Engineers (IEEE)

The Internet is an important means for communication (also of sensitive data) and even broadcasting. In the Netherlands, a vast majority of the citizens have access to high speed (broadband) Internet connection. In some countries,

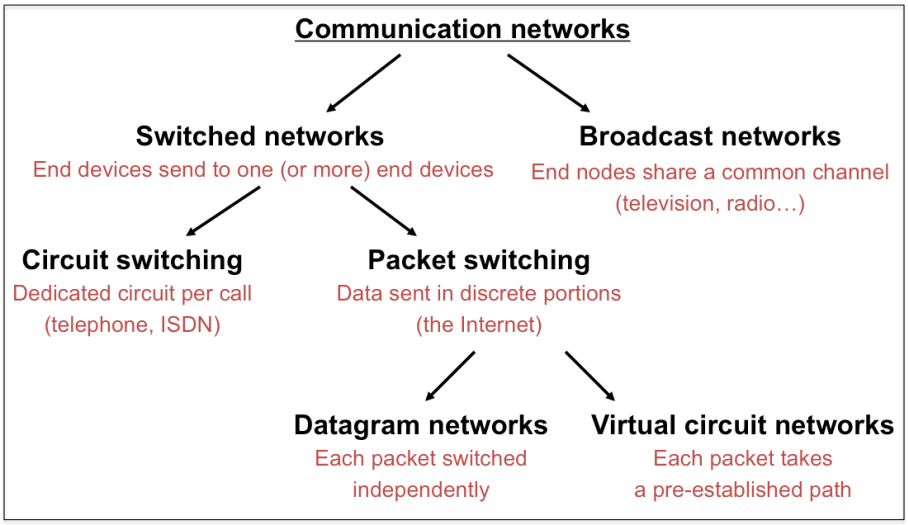


Figure 1.7: Communication network types. (Figure by Igor Radovanovic)

like Finland, citizens *legally* have a right to broadband Internet. That is, in these developed countries, high speed access to the Internet is a *civil right*, just like getting education and health care. Given this significant penetration into societies, in every country (some more strict than others), the government wants to regulate the use and the utilization of the Internet. In doing so, their goals are many. Government regulations can be, for example, for the sake of:

- fighting digital crimes (e.g. protecting intellectual property, fighting piracy)
- facilitating open market competition (e.g. to eliminate monopoly)
- practicing censorship (in some countries)

On top of these country-wide (or sometimes near-global) regulations, there can also be local regulations enforced by network administrators (e.g. institution or company regulations). Most institutions and companies would not allow BitTorrent clients to be used in their networks and they enforce this typically by monitoring network traffic and detecting "unusual" traffic behaviors. In this course, we discuss how this can be done in detail.

1.4 Network Physical Infrastructure

A taxonomy of communication networks is given in Figure 1.7.

A computer network infrastructure consists of leaf nodes that are connected to the **network core** (a sea of interconnected routers) through **access networks** as shown in Figure 1.8.

This section gives an overview of the computer network physical infrastructure as well as the two basic methods to realize data transfer over the infrastructure: *(virtual) circuit switching* and *packet switching*.

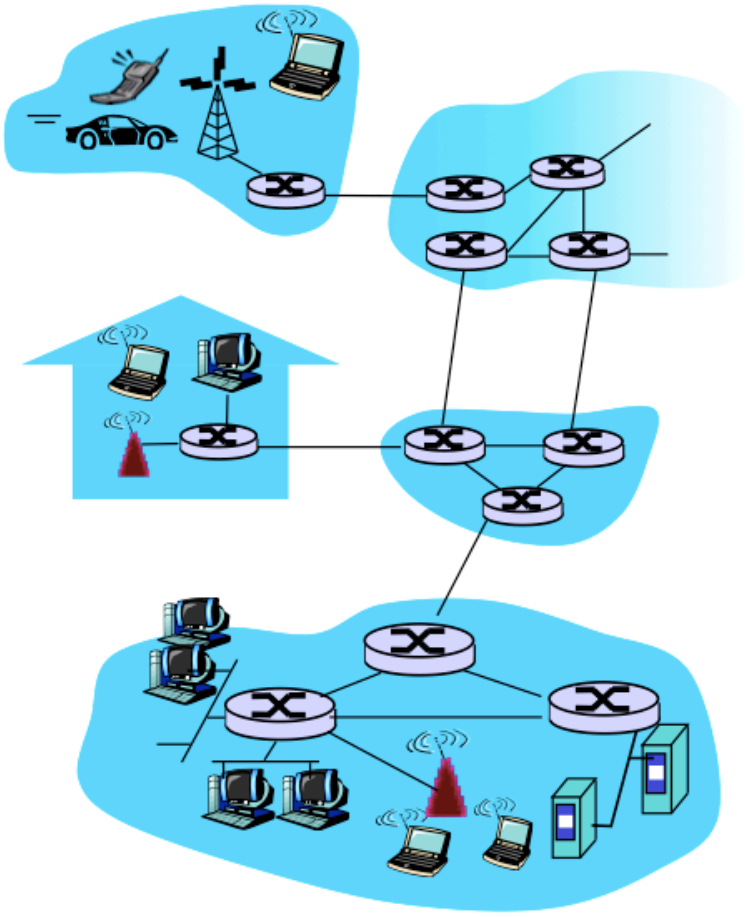


Figure 1.8: The network infrastructure consisting of end devices, access networks and network core. (Figure by Kurose and Ross)

Figure 1.9: Smart spaces are advanced computer networks where the user is in the center of all, i.e. smart space applications are there to satisfy the user. Many applications with various characteristics can be realized by devices that surround the user. These can be provided by individual devices or collaboratively.

1.4.1 End Devices and Access Networks

At the edge of a network there are networked **end devices**, also called **hosts**, where network applications reside. Examples of hosts running networked applications are **clients** and **servers**. In typical computer networks terminology, *a client is the party that initiates a connection while a server is the one that continuously awaits incoming connections.*

The importance and the structural complexity of computer networks are only growing with new concepts of '**smart X**'. Smart phones ensure people can be online whenever and wherever. Smart grid infrastructures and smart metering enable remote monitoring and control over the electricity grid, and similarly for other critical infrastructures. Other upcoming technologies such as smart cars and roads make traffic an interconnected moving and evolving network. Nowadays, there is immense research effort all around the world, especially in developed countries, on **smart spaces**, where the idea is that the space covered by a network of computers and connected **embedded systems** adapts its behavior to facilitate the goals of its user(s). There are various sizes and scopes of smart space implementations such as smart cities, smart buildings and smart homes, each containing many embedded systems.

Definition 1.4.1.1 *Embedded System*
An embedded system is a special purpose computer on hardware dedicated to that purpose. These systems range from very simple sensors to complex machines.

Embedded systems vary in their networking capabilities. The evolution of embedded systems and their use in networks over the years is illustrated in Figure 1.10.

- **Standalone embedded systems** have a standalone functionality and they do not communicate to other systems.
- **Network-aware embedded systems** allow access of some (typically limited) internal functionality from outside. They come with proprietary

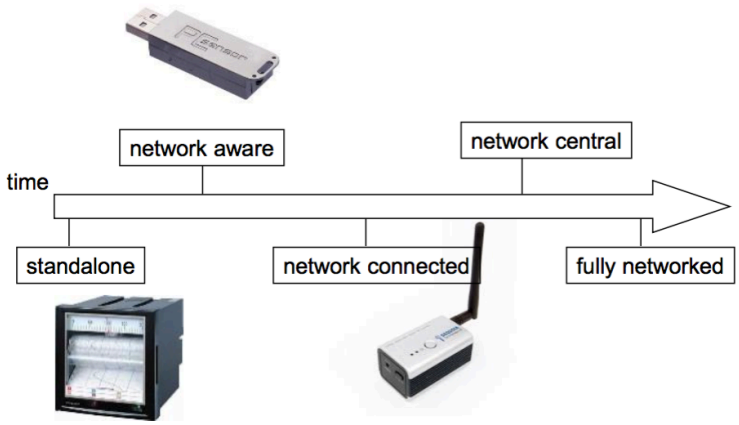


Figure 1.10: The evolution of embedded networking (Figure by Johan Lukkien).

(private) network protocols dedicated for this purpose. Simple data retrieval (sensing, diagnostics) is possible. Software updates over the network are possible, but typically not straight-forward (expert knowledge is needed).

- **Network-connected embedded systems** are 'on-line' using standard protocols that are open to the public. Networks of these typically go by the name "machine-to-machine networks". An example is a body sensor node that monitors posture of a person, warns when the posture is not right and stores the data on a remote server for the access of the physiotherapist.
- **Network-central embedded systems** have some standalone function but the design of both hardware and software aim at operation in a networked context. Examples are with many smart phone apps, television sets and intelligent lighting (e.g. Philips Hue).
- **Fully networked embedded systems** do not have a meaningful standalone function when they are disconnected from the network. These are mostly cheap devices with elementary behavior. Very low resource devices are typically fully networked. Examples are with applications of simple sensing and actuating and elementary computing.

In general, devices can be classified (as of today) as shown in Figure 1.11, where each row corresponds to a different device class.

Access Networks: An access network connects the network edge to a router in the network core. These are typically shared networks where multiple end points can connect to the (rest of the) Internet. Access networks can be of variety of sorts, e.g. mobile access networks, wireless and wired access networks for residences, businesses and institutional areas.

For many years, **Digital Subscriber Line (DSL)** over the household's telephone line and **Cable Internet** access over the household's cable television network connection have been the dominating access technologies for homes. The vast majority of broadband connections in the world are through DSL.

	Flash	RAM	Address space	Processor (type)	OS	Energy	Operation	Actively reachable	Example
A	small code memory	several bytes	<= 8bits	~100Hz	no	External, or battery + wakeup	Externally activated, simple read/write	not designed for reachability via multi-hop	RFID tag, ISO 18000-6c
B	<= 32K	Few hundreds	<=16 bits	~1Mhz TMS430	no, or simple executive	mechanical	mechanically activated, just generates some data	no; needs proxy	power switch
C	<=32K	Few hundreds	<=16 bits	~1Mhz TMS430	Contiki, TinyOS	battery	simple, fixed external behavior, needs proxy, simple sensing	duty cycled, needs proxy	simple sensor mote
D	<=32K	~10K	<=16 bits	~1Mhz TMS430	Contiki, TinyOS	battery + recharge	capable of managing most constrained IP protocols, sensing, actuating, processing	self-managed on/off behavior	Crossbow
E	<=256K	~32K	<=32 bits	~1-10Mhz ARM	Contiki, TinyOS	battery + recharge, mains	complete IP endpoint behavior, limited storage	yes	Jennic mote
F	~GB	~500Mb	32 bits	~Ghz ARM	Linux	battery + recharge, mains	full fledged embedded computer system	yes	Rasberry PI
G							phones, laptops, servers		

Figure 1.11: Device classes based on available resources (Figure by Johan Lukkien).

DSL gives dedicated connection to the central office (the gateway to the Internet core) and, therefore, provides dedicated access bandwidth to its individual customers. On the other hand, cable Internet connection is shared among neighbors, i.e. more than one household hook up to the cable line, resulting in bandwidth sharing. Although outside of peak hours cable Internet is typically faster than DSL, the effective bandwidth per household is likely to drop for cable Internet during peak hours when a lot of people are online.

	DSL	Cable	FTTH
Transmission medium	Twisted copper wire.	Coaxial cable.	Fiber optic cable.
Speed	HIGH. Especially high speed when the distance to the ISP is low. Connection is not shared with other subscribers. Better response to increasing number of online users than cable.	HIGH. Higher than DSL outside of peak hours. Distance to ISP does not change the speed. Since connection is shared, performance drops during peak hours.	VERY HIGH
Download vs Upload	Both symmetric and asymmetric versions available.	Asymmetric. Download speed is much higher than upload. Download and upload share the same channel (i.e. half-duplex).	Asymmetric. Download and upload do not share the same channel (i.e. full-duplex). High upload speed nevertheless.
Infrastructure cost	LOW	LOW	HIGH

Table 1.1: DSL vs Cable vs FTTH

Recently, **Fiber to the Home (FTTH)**, which requires additional infrastructure has finally become affordable for households. FTTH is gaining momentum with more and more districts getting fiber glass infrastructure in place. A

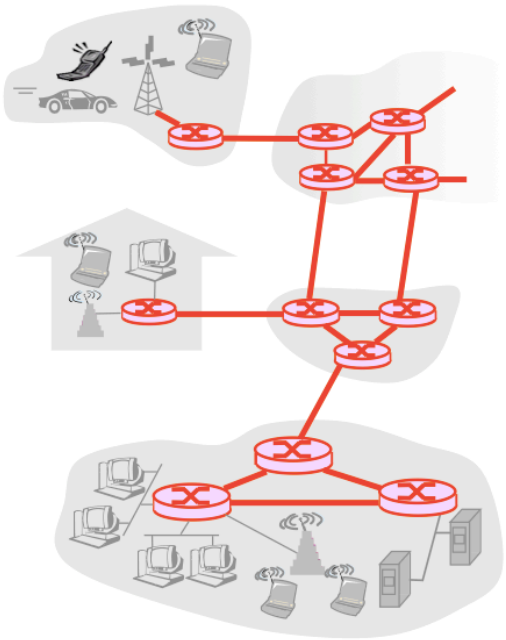


Figure 1.12: The sea of routers forming the network core (Figure by Kurose & Ross).

comparison of DSL, cable Internet and FTTH is given in Table 1.1.

1.4.2 Network Core

The network core consists of a sea of interconnected routers that can transfer data between hosts (clients, servers and peers) as shown in Figure 1.12. It facilitates data transfer between end-devices connected to different access networks. In practice, it can do this in two ways: *i)* **(virtual) circuit switching**, and *ii)* **packet switching**.

Circuit Switching: This scheme provides a dedicated (virtual) circuit per call or session. The resources on the links (e.g. link bandwidth, switch capacity) all the way on the path between the communicating entities are reserved (in both directions) for a session and these resources are not shared with any other sessions. That means, via circuit switching, there is an upper limit to the number of sessions that can be supported over a network. This gives circuit-switched sessions a guarantee on the session quality (circuit-like performance), just like the session quality guaranteed by a phone connection, to some degree at least (e.g. when you are close enough to the nearest cell tower). In order to be able to give such guarantees, your phone operator will not admit the call if the needed resources are not free. This is the same reason why it is difficult to make cellular phone calls at a concert or stadium where there are a lot of active calls using the same cell tower (we will cover wireless communication later in detail).

In a given session, the amount of resources needed can fluctuate a lot. For example, parties do not generate audio data continuously during a phone call and

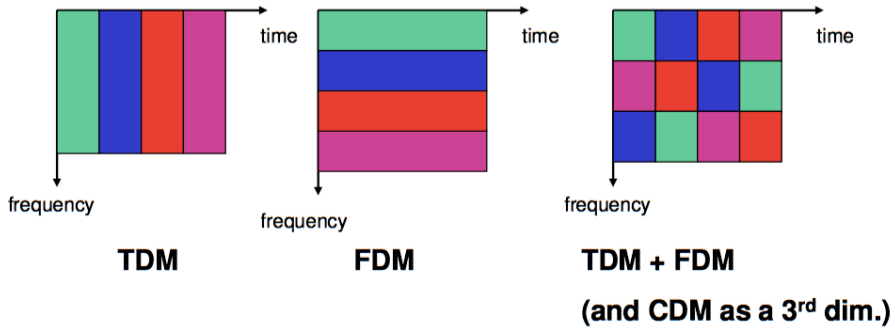


Figure 1.13: Resources allocated to four different senders (color coded) in FDM, TDM and a combination of FDM with TDM.

they pause between sentences and while listening to the other party. Resource reservation is typically done (by means of a call setup procedure) considering the maximum amount of resources needed at any given instance during a session. This in combination with not sharing resources brings the disadvantage that the resources that are not used by the current session remain idle, which is a waste.

By dividing the network resources among sessions, what circuit switching does is indeed dividing the network into logical pieces, each of which is accessible to only one session. But how can we divide a link (e.g. a wire) into logical pieces? This can be done, for example, using Frequency Division Multiplexing (FDM), Time Division Multiplexing (TDM), Code Division Multiplexing (CDM) or a combination of these.

In FDM, a different frequency subband is allocated to every session. Part of the allocated frequency band is used for receiving (downlink) while the remaining part is used for sending (uplink). In TDM, a different time slice of a (fixed) time period is allocated to every session. Resources allocated to different senders in FDM, TDM and a combination of FDM with TDM is visualized in Figure 1.13.

In CDM, every session uses a signal code which is orthogonal to all the other codes that are used by other transmitters, such that the multiplication by the session's own code will return zero for all transmissions except for the transmissions of this specific session.

Packet Switching: In packet switching, channel resources are not reserved. The available network bandwidth is not divided into logical pieces. Data packets share network resources in a **statistically multiplexed** manner and a packet uses the entire link bandwidth when it is put on the link. This is illustrated by Figure 1.14.

Since resources are used as needed, idling of channel resources is not the case as opposed to circuit switched networks. However, different packets compete for channel resources. In case there is too much demand, i.e. too many packets are sent in a short time (e.g. by many senders), the available bandwidth may not be sufficient, causing the phenomenon known as network congestion.

In packet switching, packets move one hop (a single link) at a time. This is known as the **store and forward behavior** and means that the entire packet

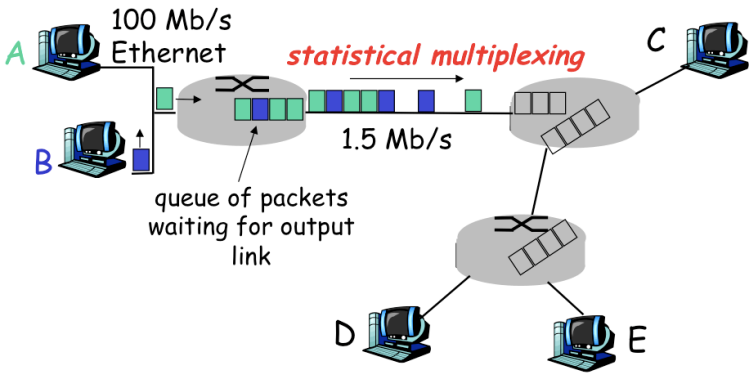


Figure 1.14: Statistical multiplexing illustrated. (Figure by Kurose and Ross)

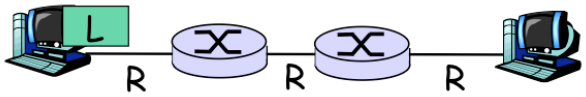


Figure 1.15: Store and forward behavior example with 2 routers between the sender and the receiver. (Figure by Kurose and Ross)

must arrive at the router before it can be transmitted on the next link. It takes L/R seconds to transmit (push out) packet of size L bits onto a link at a rate of R bps (bits per second). For the example given in Figure 1.15, the transmission delay experienced by the packet is 3L/R seconds (i.e. L/R for the transmission from the source, and 2L/R for the transmissions from the routers).

Packet switching allows more users to use the network at the same time, i.e. there is no call admission process. This comes at the expense of losing quality guarantees (see the exercise at the end of the chapter).

As given by the taxonomy of Figure 1.7, it is also possible to realize **virtual circuits** using packet switching. The term 'virtual' implies that even though the connection is packet switched (i.e. each packet uses entire channel resources), the multiplexing of packets can be done in such a way to provide circuit-like guarantees for selected sessions. We will come back to this topic later.

1.5 *The* Internet: Today

Given the general definition of a computer network that we provided in Section 1.1.2, what is it that makes ***the*** *Internet* unique? We provide here again two definitions of the Internet; one referring to its physical structure and one referring to its logical functionality.

Physically The Internet, as the largest computer network of our time, is a global public network of computer networks. It uses globally accepted (some even standardized) communication **protocols** that govern sending and receiving messages between billions of end-devices and millions of (sub)networks.

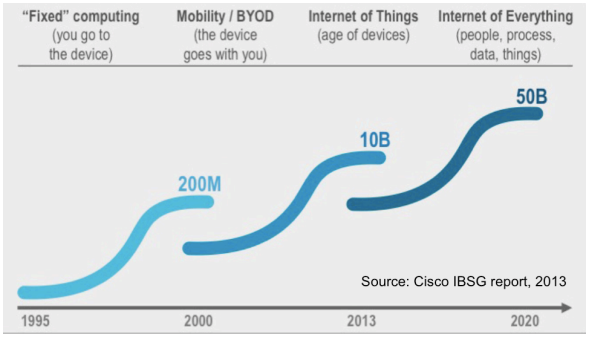


Figure 1.16: The number of "things" that are connected to the Internet will be 50 billion by year 2020 according Cisco Internet Business Solutions Group (IBSG) report. (Figure by Cisco)

Logically The Internet is a global facility that provides communication services to network applications, many of which are very well-known such as e-mail, world-wide web (www), online games, voice over IP (VoIP), Skype, Netflix and many more. It is a means for data delivery from a source to a destination, where sources and destinations can be at different abstraction levels (e.g. device versus process communication).

Just try to think of a world without the Internet. In today's world, this is almost unimaginable. Vital services of the government and the business sector depend on the Internet. Not to mention that most people would be very upset by its absence, since being "always on(line)" is nowadays crucial to many people.

> *"Internet addiction disorder (IAD) now more commonly called problematic Internet use (PIU), compulsive Internet use (CIU) ... or iDisorder, refers to excessive computer use which interferes with daily life."*
>
> Wikipedia

There is an ever increasing trend in the number of devices that are connected to the Internet, as depicted in Figure 1.16.

1.6 Internet of Things (IoT): Tomorrow

Recent developments in the technology of networks, newly introduced protocols suitable for low capacity devices[4] in combination with miniaturization of hardware make it possible to extend the reach of the Internet to fully networked systems that can be embedded in the so called "things". In the *Internet of Things (IoT)*, machine-to-machine communications (not directly involving any humans) will be responsible for the vast majority of Internet packet traffic. Hence, the two factors that separate the IoT from the Internet as we know today are: i) IoT aims to connect networked embedded systems of all types to the Internet, and ii) IoT scales to many ($>> 1$) networked devices (things) per human.

[4]Consider device classes A, B, C and D.

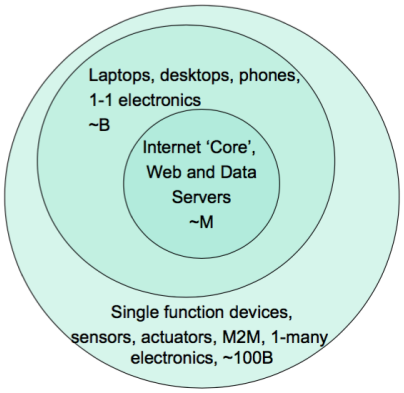


Figure 1.17: IoT devices and scale. (Figure by Johan Lukkien)

Definition 1.6.0.1 *Things*

'Things' in the context of the IoT are constrained devices, which are connected to constrained networks (typically deriving from node constraints) but then, united with 'regular' Internet services and broadband networks.

Things and their local networks have the following constraints:

- low memory: static (flash) and dynamic (RAM)
- low processing power: number of instructions per second
- low available energy: often battery powered devices
- low accessibility: devices that go to sleep to save energy (duty cycling)
- low throughput: number of bits communicated per second
- high and fluctuating packet loss rates
- asymmetric links: download capacity is more than upload
- very small packet size: an IP packet may not be delivered in one go.
- limited group communication primitives: addressing multiple devices at once is difficult.

Physically The IoT is the Internet plus an extension of the Internet into the physical world surrounding us, which is monitored and affected by *things*: constrained devices with limited memory, processing power, energy and accessibility. *Things* are connected through Internet-enabled constrained networks (deriving from device constraints) but then united with fast networks and regular Internet services.

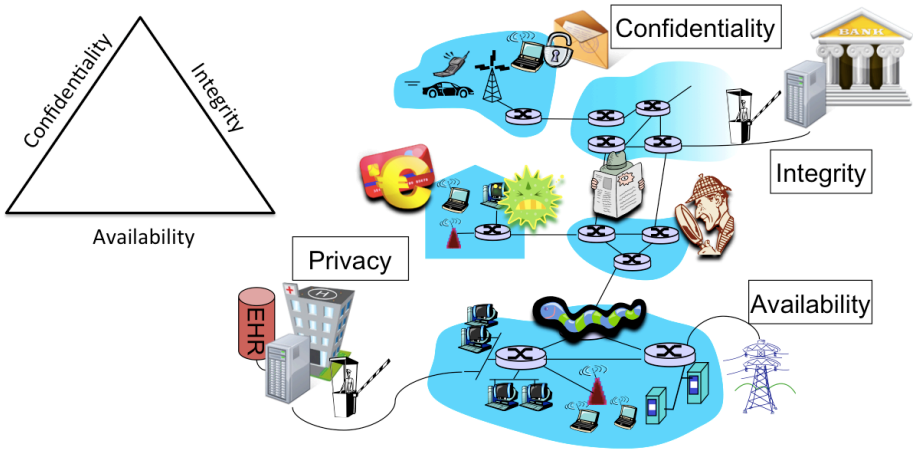


Figure 1.18: The C-I-A triad and security attributes in a network context.

Logically The IoT stands for the vision of the Internet of tomorrow. It is a global facility that extends the reach of distributed applications to billions of resource-poor devices. The IoT brings endless possibilities for innovative scenarios (e.g. smart homes, smart health care, smart buildings, smart cities), characterized by the fact that distributed IoT applications consist of collaborating services running on many distinct devices.

As suggested by this definition, the IoT extends Internet communication to billions of endpoints and reaches into the physical world, gathering incredible amounts of data about states and events of processes, objects, people and the physical environment itself. It enhances coordination and algorithms to use this knowledge as soon as possible in feedback cycles and new applications, thus connecting the physical and the Internet worlds. In order to realize this vision, the corresponding Internet technologies (hardware and protocols) need to have their simplified and standardized IoT counterparts. Even though *things* are very limited in their processing and storage capacities, the IoT provides processing and storage capacities to analyze such data by defining places where these resources are available: distributed decentralized computing and storage, remote cloud computing and storage, and solutions in between.

1.7 Network Security

We entrust networks with our social life (e.g. social networks), our money (e.g. Internet banking), our health (e.g. electronic health records) and more. The high value combined with the fact that networks inherently connect multiple parties, each with their own interests, means that they are a natural target for attackers. Ensuring security of networks is thus of paramount importance. Yet what does that exactly mean? Which security goals of which stakeholders are important to achieve?

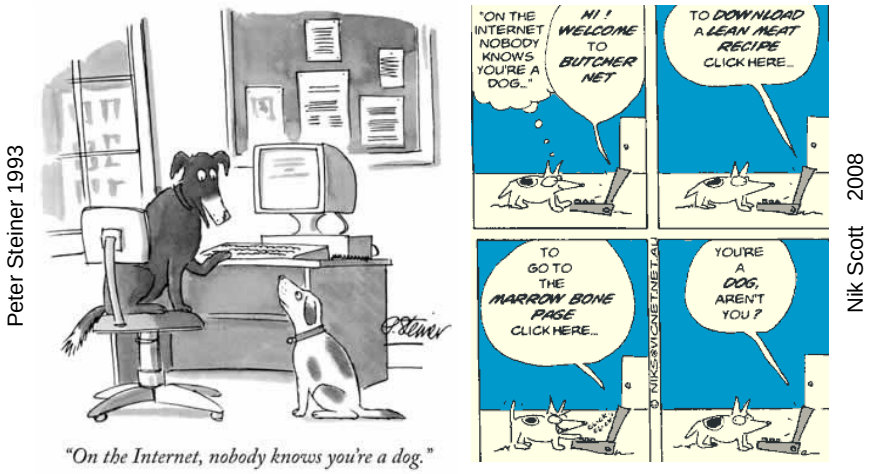


Figure 1.19: Two views on privacy on the internet

1.7.1 Security and Network Security Goals

Networked systems (simple apps, complex networks, complete IT infrastructures) operate in environments involving different interconnected parties each with their own agenda (goals), which may not match with the goals of other parties of the system as whole. As such, it is essential to also consider the security requirements of systems (i.e. what should not go wrong), not only their functional requirements (i.e. what the systems should achieve).

"Is your system secure?" What does this question actually mean; does it mean that nobody but you can use it; can throw it out the window; can keep you from using it...? Security requirements are expressed in terms of *security attributes* that express goals that one may want to achieve to call a system 'secure'. The most commonly used and widely accepted security attributes are *Confidentiality*, i.e. 'my information stays secret', *Integrity*, i.e. 'my information stays correct', and *Availability*, i.e. 'I can get at my information' (sometimes called the C-I-A triad). Of course these concepts can also refer to resources or system aspects other than just 'information'.

In addition to Confidentiality, Integrity and Availability ('C-I-A') other security attributes are sometimes formulated. Closely related but not usually called a security attribute is *Privacy*, i.e. 'information about me is not misused'. Note the difference between Confidentiality and Privacy: where confidentiality requires data that you possess to remain secret, privacy deals with data about you that may be in the hands of others. While 'who gets the data' is a key question in confidentiality, the purpose for which data is used is a key ingredient for privacy.

Other examples of security attributes are: *Authenticity*, i.e. 'is this information authentic (i.e. of undisputed origin)', *Non-repudiation*, i.e. 'is this information undeniable' and *Accountability*, i.e. 'is the information provider accountable (can we punish the provider if the information is incorrect)'. Authenticity is dif-

ferent from integrity in that it focuses on data coming from the 'correct' source rather than on data not being changed along the way. A signature on a contract would be an example of a way to achieve non-repudiation; you cannot later deny agreeing to the conditions in the contract. The relation between accountability and non-repudiation is similar to that between integrity and authenticity; non-repudiation can be an important part of achieving accountability but is by itself not sufficient.

The security requirements together with the *security policies* of a system tell you what attributes should be achieved when (in which context). The requirements will typically say what security attributes should be achieved by which components and/or for what type of resources (e.g. confidential database entries should only be readable by user with the right clearance). Security policies detail this e.g. by stating what type of data is confidential and what (types of) users have clearance. Security requirements are an integral part of the design of the system while changes of policies is typically taken into account and should not invalidate the design. Note, however, that the term security policy is widely used and the exact interpretation varies. It could be a high level textual description meant to be understood and applied by human beings, e.g. "all personal identifiable information must only be read when needed to provide a service" to low level computer readable information e.g. "drwxr-xr-x"[5]. Translating high level policies into a systems design along with low level policies is an important step of creating a secure system.

The exact meaning of a security policy can be given within a *security model*; a (formal) framework to express and interpret policies. For example, the Unix file permission given above can be interpreted as a relation between *Users*, *Groups*, *Objects* and *Permissions*: An object (e.g. a directory) has an owner user and a group (an additional part of the security policy) and the owner of the object has read, write and execute permission, while members of the group as well as other users have only read and execute permission.

1.7.2 Threats

The security attributes of the system may be at *risk* from several types of *threats*. Besides the usual problem such as program errors and system failures, security also needs to address malicious entities, which are specifically trying to break the system. This is very challenging; every day seems to bring new security incidents where attackers are able to exploit (previously unknown) security weaknesses. Although this may give a skewed perspective (a system remaining secure yet another day will not make the news), it does show the importance of applying the right mechanisms for securing your system.

To decide what the right mechanisms are to achieve the security requirements of the system, we need to know whom we want to protect against. Protecting information in a database from an outsider requires different solutions than protecting it from the database administrator. We thus need an *attacker model*. This attacker model captures the capabilities and possibly the intentions of an attacker. For example, in a network setting we may distinguish between attackers that can only listen in (eavesdrop) and those that can block and/or modify communication.

[5] A Unix-style file "read-write-execute" permission setting.

Attacker models can be general, e.g. IBM's classification of attackers into three categories (clever outsiders, knowledgeable insiders and funded organizations); or formal, e.g. those used in analysis of cryptographic algorithms (e.g. Chosen-Plaintext-Attack (CPA) where the attacker is able to get encryptions of plain text she has chosen). Any security analysis will need both the security goals (attributes/policy) and the attacker model. Sometimes these are left implicit but they remain key ingredients; the question 'is this system secure?' has no meaning without them. Not properly considering them is a common cause of security problems.

1.7.3 Security Engineering

A chain is no stronger than its weakest link. This is also the case for the security of a system. Consider for example the following aspects of a system and some potential issues.

Design There is no hope of having a secure system if the system design does not address security goals or worse has inherent features/goals that imply security problems. As an example consider the Windows Meta File (WMF) where arbitrary code execution, a clear security risk, is a design feature.

As another example one can consider the Internet; initially the Internet linked a group of trusted systems. Security goals that are very important now were thus not under consideration in its design, e.g. no protection of content, any computer can claim to have an IP, no authentication of DNS, etc. Of course there are currently security mechanisms (IPsec, HTTPS, etc.) that try to remedy this but 'add on security' is always problematic - security needs to be considered from the start.

Software quality A perfect design does not help if the implementation is flawed. Often security issues are caused by software bugs with buffer overflow vulnerabilities being one of the major issues. In buffer overflow attacks input from an untrusted source is written into a buffer without the bounds of the buffer being checked. This causes the untrusted data to be written to places it is not supposed to go; it may overwrite a return address on the stack, causing a jump to an attacker selected location at the end of the current routine.

The problem of software bugs is not solved easily; e.g. an unsolved buffer overflow vulnerability was reported in Windows 7 and in January 2011 Microsoft shipped fixes for 22 vulnerabilities. The 'heartbleed bug' (see Figure 1.20) is a recent example of a software flaw related security incident with wide media coverage. Note that software and systems evolve. It is not the case that each round of patching brings us closer to a final secure and bug free system.

Security Tool Selection Choose your crypto well, especially if you are a mafia boss. From a news article: "...He apparently wrote notes to his henchmen using a modified form of the Caesar Cipher, which was easily cracked by the police and resulted in further arrests of collaborators..." Clearly here the selected security tool was grossly insufficient to reach the security goal.

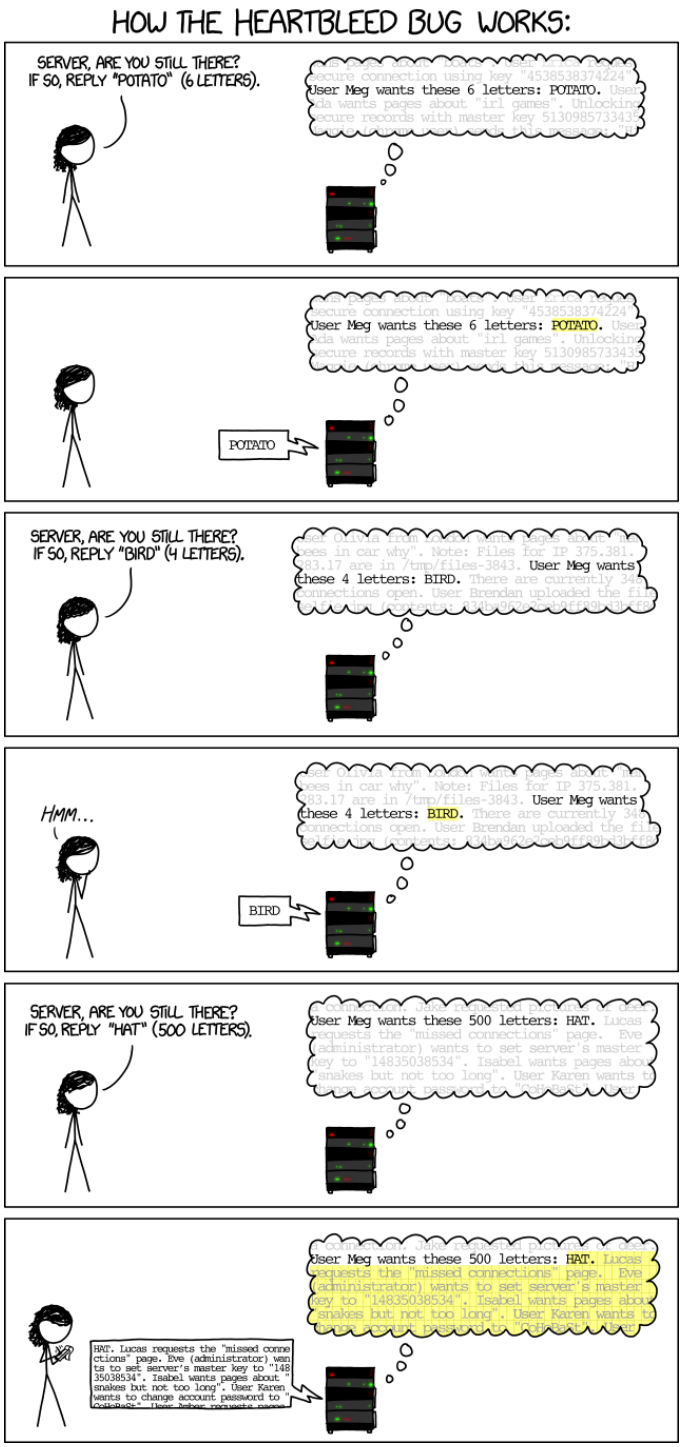


Figure 1.20: Heartbleed explained (xkcd.com)

This is an extreme example but often inappropriate security tools are used or tools are used well past their 'best before/replace by' date such as the hash function MD5, which has been known to be vulnerable for a long time but is only slowly being phased out. Using 'home-made' crypto solutions instead of tried and proven standard algorithms would also fit in this category. A good practice is to leave design of crypto to the experts; obscurity of a design is not a good replacement for their experience and expertise.

System usage Even a perfectly designed and implemented security architecture (should one ever be created) is of no help if it is not used correctly. USB data sticks that offer encryption of their content are readily available and company policy may state that the encryption of such sticks should be used. However, if the user does not enable this feature this is all for nothing.

Users have different priorities; e.g. ease of use; and many do not use security features or will even try to work around them if they interfere with what they are trying to do.

Of course these are only examples and there are many more aspects of a system where a weak link in the security chain may occur. The key points are that one needs to consider the system *as a whole* and consider security *from the start*.

We have already seen some security tools (means) above and later we will try to add key tools to this toolbox, focusing on network scenarios. Cryptography is an important part of this toolbox. However recall that security tools by themselves do not make the system secure. A common claim 'the data is secure because it is encrypted' is by itself meaningless and may even indicate that the security goals and the attacker model have not been sufficiently considered. For instance, encryption offers no protection against inside attackers who have access to the key. A good security design determines what security tools need to be employed where and when, considering the security requirements and the effects (including trade-offs) different tools have on these requirements.

Trade-offs

> *"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards."*
>
> E. Spafford

Such a system may be secure but not very useful. (Actually it may not be secure at all - Which security attribute is clearly not satisfied? - without the security goals we cannot answer this question...) There is often a clear trade-off between security and usability (why do I need to remember that password...), performance (e.g. using encryption adds computation time) and costs (e.g. replacing pin cards and readers by smart card enabled versions). There is also a trade-off between different security attributes e.g. confidentiality and availability. We have to be able to answer the question: Which trade-offs are worthwhile; e.g. how much security do we gain for the performance we give up?

Why does security often not get the attention it needs? For one; if it's good you do not see it. Would you pay 50 Euro more for a television if it was more secure? Does your answer depend on 'how much' more secure? It is also hard to quantify security. You can say that a 'product is 2 times faster' and convince every consumer with some notion of why and how much better the product is, even though this statement is usually much more complex than it seems. However, what does 'this product is 2 times more secure' mean? There are many discussions on which product is more secure, e.g. comparisons between Windows and Linux, Firefox and Windows Explorer, Mac and PC, etc. Claims are supported by quoting the number of bugs/vulnerabilities reported, the number of security incidents, etc. But how well do any of these really reflect the overall 'security' of a system. Thinking back to the earlier discussion about what is 'security of a system' one can see that no single number could really adequately capture this. Still, what quantification is possible? If we try to focus our attention on a single aspect of security and a single application area, one may be able to give some numbers that make sense (just remember that, the more general the statement the less objective a score is likely to be).

For cryptographic primitives one can look at the (computational) cost of breaking a system. This is often expressed by the *entropy* that it offers in a given setting, e.g. 'this crypto system offers 80-bits of security' reflects that the amount of computation needed to break it is similar to brute-forcing an 80 bits key, i.e. trying 2^{80} different possibilities. This is generalized to a measure for security of systems by considering the cost (computational or otherwise) of breaking the system's security; e.g. it would take 2 years and a budget of 10 million euros to break this system (i.e. violate a specific security goal of the system).

For web applications several security metrics have been defined by checking for common security issues and assigning a risk to each of them. For example, the CCWAPSS common criteria for web application security scoring [8] computes a score based on a list of eleven criteria. Each criteria has to be checked (rating the web service on a scale from 1 to 3 for each item) and assigned a risk level based on the difficulty and impact of an attack.

Security Requirement Engineering

As already mentioned several times (and as will be repeated later in more detail), to really evaluate the security of a system you have to consider it as a whole, know the security goals and the potential threats against these goals. To gather these we need to perform *Security Requirement Engineering*. Throughout the design, implementation, deployment and use of a system we should consider the requirements that the users will have from the system and how attackers will try to exploit the system. Based on this we can come up with and/or evaluate a security design, which combines several security solutions to achieve the best possible trade-offs.

Here we shortly cover one example security requirement approach. Other approaches may work just as well, what is important is that the security requirements are considered throughout in a structured and consistent way.

Identify actors and goals. The first step in gathering the requirements is determining the stakeholders and their interests. The stakeholders are those parties with a legitimate interest in the system that we are designing.

Their interest and goals thus have to be considered (though not necessarily completely reached - we may need to make trade-offs between the different goals of the participants).

The stakeholders and their interests become the initial actors and goals in the requirements gathering process. If an agent has the right capabilities, it may *adopt* a goal, i.e. take responsibility to achieve it. If an agent does not adopt the goal it may be delegated to other agents (either existing or new) or be split into new sub goals. Agents do not work in isolation; agents and their goals may depend on/interact with each other. These dependencies should be identified and could lead to new goals and/or agents. They also lead to potential vulnerabilities, e.g. when agents' goals conflict.

So far the process matches a typical functional requirement engineering approach. In order to deal with security requirements we also need to consider attackers and possible attacks on the system.

Identify attackers, vulnerabilities and attacks. Outsiders may try to attack our system and they need to be considered along with their goals. However, also the risk of attacks by insiders needs to be accounted for. Each agent in the system could potentially become an attacker, using its capabilities and place in the system to reach their goals at the expense of the goals of other agents. Both type of attackers are modeled as agents in the system but with malicious intent as their goal.

Based on vulnerabilities and the malicious intent of attacker agents we identify potential attacks and assign countermeasures to protect against such attacks. The countermeasures themselves may lead to new actors/-goals and/or open the possibility for new attacks which need to be considered. Refinement of the system continues until all goals have been assigned, dependencies taken into account, and vulnerabilities addressed.

1.8 Summary

The goal of this chapter was to introduce the most basic and fundamental concepts of computer networks and network security, as well as the motivations for their existence. You now know the principle nuts and bolts of a computer network, the idea behind network protocols and protocol layering. A simple overview of the Internet and the Internet protocol stack have been provided in our discussion, together with an outlook into the future of the Internet dominated by machine-to-machine communication, i.e. the Internet of Things. After our security discussion, you will never look at the word 'secure' in the same way again: Whenever you encounter 'secure' always think - what set of security requirements (which security attributes for which resources) are really meant by 'secure' (what are the security policy and model) and what type of attacker is considered (what is the attacker model). The notions introduced here will return in more detail in the chapters that follow.

1.8.1 Literature

Other required reading

- Article titled "A goal oriented approach for modeling and analyzing security trade-offs" [15].
  - `http://www.cs.utoronto.ca/~gelahi/GEER-ER07.pdf`
- CCWASPSS white paper [8] (provides a basic method for quantifying website security).
  - `https://dl.packetstormsecurity.net/papers/web/ccwapss_1.1.pdf`

Suggested reading

- Section 8.1 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Sec 8.1].
- Chapter 1 of the Security Engineering book by Anderson [3, Ch 1]. The book provides a nice high level overview and in particular the first chapter introduces the concept of security engineering.
- The OWASP website www.owasp.org. The website provides articles on network security and is a good index for many security terms, attacks, tools, etc. In particular 'the OWASP top 10' is a well known overview of important security threats.

1.9 Homework Exercises

1. What type of embedded system corresponds to the ants in the network analogy of the ant colony?
2. Name two regulations on the Internet by the government of your home country.
3. Name one disadvantage of the store and forward behavior. (Specifically of store and forward, not e.g. of packet switching vs circuit switching. Note that store and forward behavior could in theory also be used within a circuit switching network.)
4. Consider a prescription for medicine from your GP (General Practitioner) that is filled by a pharmacy sending the medicine to your home. For each of the following security violations, indicate the main/primary security attribute (including privacy) that is violated.
   - Another patient reads the prescription.
   - You change the prescription from 1 to 10 pills.
   - The prescription is signed by someone who is not an MD (Doctor of Medicine).
   - The GP denies signing the prescription.

- The pharmacy sends you your prescription at home but also starts sending you advertisements for products.
- Your medicine is lost in the mail.

5. Consider again the prescription for medicine from your GP that is filled by a pharmacy sending the medicine to your home mentioned in the review questions. Create a (textual) security policy for this scenario which describes which security properties(attributes) must be satisfied when. Indicate the involved security attributes for different statements in your policy.

6. An online music store allows its members to listen to music with embedded ads for free and to download music without ads for a fee. Members can also recommend songs to other members and get a free ringtone if at least five people listen to a song based on this recommendation. Do the first steps in a basic 'security requirements engineering' for this scenario: identify actors, their interests and interdependencies. Also find attackers and their goals. (As we discuss security tools in later chapters, you can enhance this design by extending it with potential countermeasures.)

7. Find a security related news article and analyze it (collect some background information when needed).

   - What is the security issue in this article? Is it related to availability, confidentiality, integrity?
   - For a security incident; what was the failure, why did it occur, how could it have been prevented, how should it be solved?
   - For a solution/technology; what problem is solved, how can it be used, will it work?
   - For an opinion/analysis/...; do you agree, what are possible other/-counter arguments?
   - For a more general article; what is the issue you have identified, did the article address this issue? How well is the issue described, does the article get the key points correct? Did it miss any issues?
   - Is the article biased/one-sided? Does it consider the issue from the perspective of different stakeholders?
   - Do you agree with the conclusion of the article?

Chapter 2

Network Protocols and Protocol Layers

In this chapter, we try to answer the following questions:

- What is a protocol? How are protocols used in computer networks?
- What does protocol layering mean? How is it useful?
- What are well-known layered network protocol stacks?
- What are the factors that affect network performance?

2.1 Network Protocols

Protocols determine the way of operation and define the procedure and rules that are governing. Remember the example of the ant colony that we gave earlier. Each member of the ant colony implements a simple protocol in searching for food. The following three rules apply, where the timing of applying these rules are given in **bold fonts**.

1. **Always** leave a trail of pheromone wherever you go.
2. **Whenever** you come across existing trails that are stronger, join them.
3. **Once** you find food, follow the pheromone trail back to the nest.

Figure 2.1 shows a typical purchase protocol defining the way we buy food at the cafeteria. An important observation is that the cashier starts serving the user only after the user has waited for her turn in the queue. Some sort of greeting ("Hello" in this example message flow diagram) starts their interaction and another greeting ("Bye" in this example) ends their interaction. After this point the cashier can start serving the next customer.

Note that the figure shows only the "relevant" interactions and skips details such as "the customer takes out her wallet". In general, the description of a protocol should be as detailed as needed. Details that do not matter are skipped. For example, a certain customer may not carry a wallet and just keep

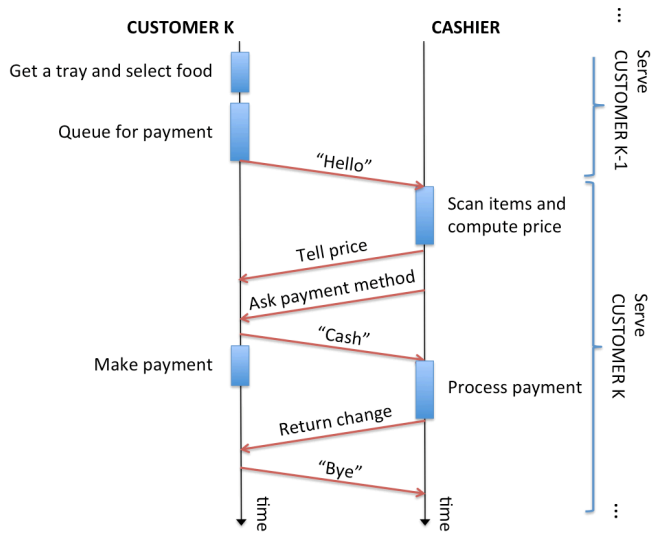


Figure 2.1: A customer-cashier protocol at a cafeteria.

the money in his pocket. Protocols are detailed further until all ambiguities regarding the operation are completely removed.

As another example; there is social protocol in every culture. In the Netherlands, during a conversation, one has to wait until the other person stops speaking. Interrupting others when they are speaking is perceived as rude behavior. Likewise, one who is speaking should give others the opportunity to respond. This is how you can get a conversation going. Looking at your mobile phone while someone is talking to you is impolite.

The governance of computer networks by network protocols is not any different. They define exactly how the interaction sequence between entities and the corresponding actions must be in order to achieve some networking goal. Networking goals are of many sorts and at different levels of a computer network. For example, a networked application's goal is to exchange application messages between remote processes and using this for the application's main purpose (e.g. file sharing). The networking goal of routing is to deliver messages end-to-end from a source device to a destination device.

Definition 2.1.0.1 *Network Protocol*

A network protocol is a predefined set of messaging rules and message structures that the communicating entities in a computer network must follow.

2.2 Protocol Layering

Computer networks are diverse and complex. In fact, the complexity is a direct result of diversity. The diversity stems from the need for including end devices (hosts) that are

- of many sorts, models and capacities;

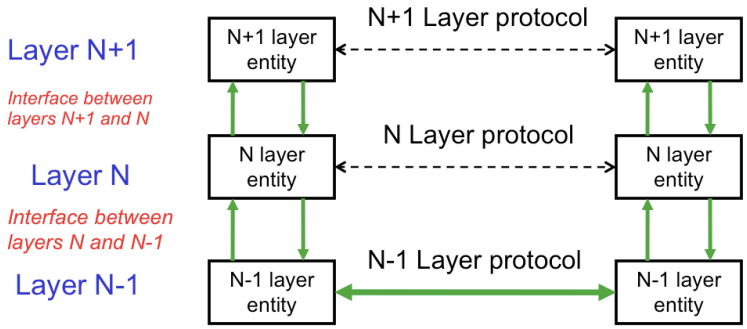


Figure 2.2: Protocol layering: Layers above use the services of the layers below.

- from many different vendors;
- running different applications;
- with different operating systems and middleware;
- over many types of access networks and link technologies;
- with different usage requirements (e.g. mobile versus immobile).

Large networks like the global Internet have to deal with the complexity at the corresponding large scale. For example, a message created by an electric car about its usage patterns while it is on the move should be able to find its way to a database implemented in a large server rack of the car company somewhere across the globe.

A **protocol layer** refers to a group of related functions that are performed in a given level in a hierarchy (of groups of related functions). **Protocol layering** divides complex network protocols into simple network protocols. Protocol layers above make use of the functions of the layers below as depicted in Figure 2.2. A set of horizontal protocols together with the interactions between the different layers define a **layered protocol stack**.

As an analogy, consider the following list of horizontal protocols that govern a restaurant environment and the way of operation there.

- A seat management protocol: Defines the way seating is realized (e.g. first-come-first-serve or reservation system).
- A table set-up protocol: Defines the operation of the waiters for setting up cutlery and plates.
- A table cleaning protocol: Defines the operation of the cleaning personnel for cleaning the tables.

Note that the *table set-up protocol* needs the services of the *table cleaning protocol* (tables are first cleaned and then set up). Similarly, the *seat management protocol* needs the services of the *table set-up protocol* (table needs to be set-up before it can be given to customers). One thing to notice here is that each of these individual protocols can be modified independently, without affecting

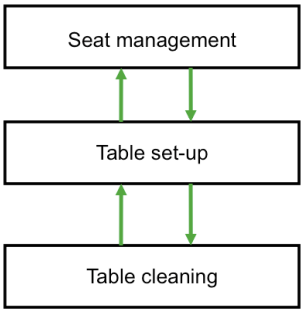


Figure 2.3: Protocol stack consisting of restaurant protocols.

the other protocols. For example, the cleaning personnel can change and they can do cleaning differently. However, this changes nothing in the table set-up protocol. The corresponding protocol stack is shown in Figure 2.3.

It is also possible to make a *vertical (monolithic) protocol design* without using protocol layering; i.e. a single protocol that governs communication. In vertical protocol design, each new application has to be re-implemented per device type, per operating system, per access link technology.

While dealing with complexity, layering brings overhead, i.e. the performance of a layered protocol stack is sub-optimal (extra headers, extra processing, etc.). For that reason, for applications with simple communication, where resource management and optimization is the main concern and where there is a dedicated (embedded) hardware platform, a vertical design may be preferable. For example, consider sending a robot to Mars. The software of the robot has to be optimized such that a lot of functionality can fit in a small volume and weight. In the past vertical protocol design for embedded wireless sensor devices was the common practice due to severe lack of resources. It is not anymore since current devices in classes D[1] and above are able to manage lightweight Internet protocol stacks.

Horizontal protocol design and protocol layering are our choices when simplicity and convenience are the main concerns instead of resource optimization. Internet would not have been possible without protocol layering.

2.3 Network Protocol Stacks

A protocol stack is a set of layered horizontal protocols. Figure 2.3 depicts an example protocol stack for the restaurant environment. A network protocol stack does the same in network environments. Each layer uses the services of the layer below, and it provides its services to the layer above.

Protocol layering and protocol stacks enable MANY applications on top of MANY physical and link layer technologies. This is shown in Figure 2.4.

There are many protocol stacks for many different purposes. In the following subsections we are going to look at two widely known protocol stacks: i) the Internet protocol stack, and ii) the OSI reference model. The protocol stacks that are widely accepted for wireless sensor networks, vehicular communication

[1]See device classes in Chapter 1.

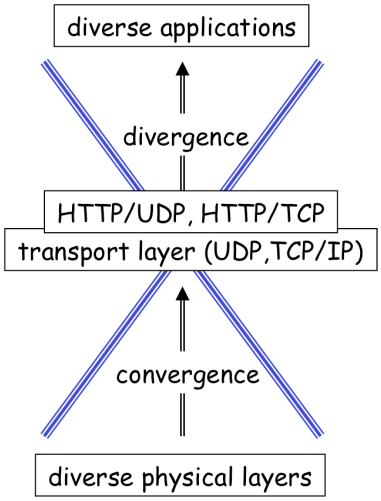


Figure 2.4: Network protocol stacks allow convergence from a large set of physical layers to protocols and divergence from protocols towards a huge set of applications. (Figure by Johan Lukkien).

networks and others will look possibly more complex and detailed. The "good" news is, we will not go into very complex protocols in this course. However, the fundamental knowledge that you acquire in the course should enable you to study any protocol stack and understand it easily.

2.3.1 Internet (TCP/IP) Model

The Internet protocol stack shown in Figure 2.5 is often referred to as the TCP/IP protocol stack, after the two dominating protocols TCP (transport layer) and IP (network layer).

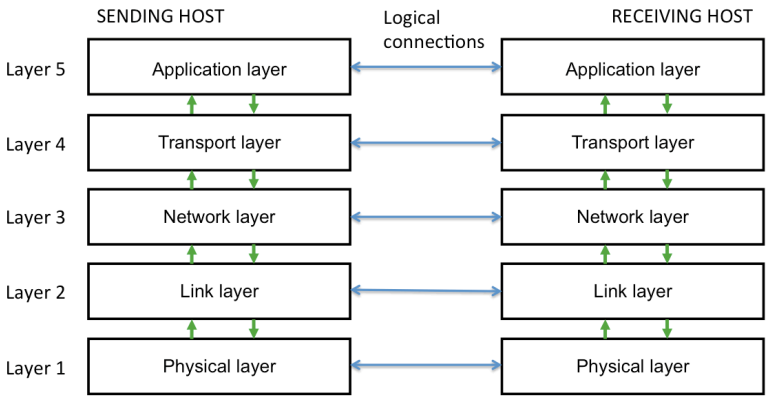


Figure 2.5: The Internet protocol stack.

(5) Application Layer: There are millions of Internet applications. There can also be many instances of a certain type of application. Just think about how many alternatives you have for a web browser. It is not possible (nor desirable) to standardize applications and how they utilize communication. Otherwise all web browsers would behave the same. Therefore, the applications themselves and how they use the information that they communicate are not concerns of the application layer. What the application layer (layer 5) provides instead is a well-defined protocol interface and message formats to applications running on top of them. The world-wide web is enabled because every web browser developer makes use of such interface of the underlying application protocol and every web browser communicates to web servers using the same rules and message formats (i.e. the HTTP protocol that we will cover later).

(4) Transport Layer: The transport layer (layer 4) is responsible for transport of application layer messages between (remote) processes. The transport layer protocol must keep track of application data from several processes and package those in transport layer packets called **segments** with certain labeling. It relies on the network layer (layer 3) for delivery of segments between the end-devices containing the processes. The receiver side of the transport layer protocol delivers the messages within those segments to the corresponding receiver processes using the segment labeling. In addition to process-to-process delivery, transport layer services also include reliable data transfer, flow control (match speed between sender and receiver), and congestion control (keep the network load below capacity).

(3) Network Layer: The network layer (layer 3) is responsible for end-to-end packet delivery between two devices (from the original source to a destination) that are attached to different (sub)networks. A network layer packet is called a **datagram** and contains a transport layer segment as its payload. Other important services of the network layer are logical addressing, routing, error handling and quality of service control. In their end-to-end path from source to destination, datagrams pass through many routers and links between routers. The network layer relies on the services of the link layer (layer 2) for delivery on a single link, e.g. between two consecutive routers.

(2) Data Link Layer: A link layer (layer 2) packet is called a **frame**. It contains a network layer datagram, which it carries to the next node (either a router or the destination host) on the end-to-end route. Link layer services also include physical addressing, medium access control (MAC), link flow control and error handling. A link can be wired or wireless. The set of all devices interconnected via the link layer is is called a **network** or a **subnet**.

(1) Physical Layer: The physical layer (layer 1) operates on individual bits and it is responsible for moving bits (actually, electrical, optical, magnetic or radio signals representing bits) from one node to the next. The original design of the TCP/IP protocol suite did not have this layer and it was added later.

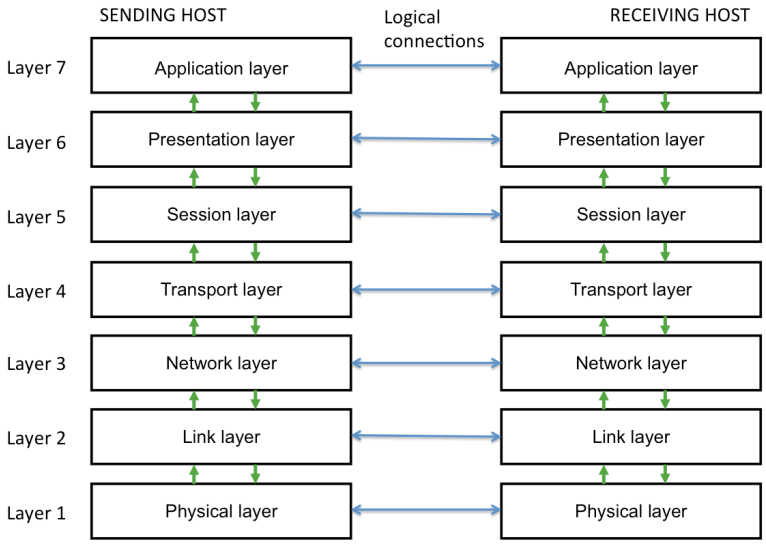


Figure 2.6: The OSI reference model.

2.3.2 OSI Reference Model

The design of the Internet model is older than the Open Systems Interconnection (OSI) model developed by the International Organization for Standardization (ISO) and it is widely used in practice.

The OSI model shown in Figure 2.6, however, provides a generically accepted conceptual reference model of a layered protocol stack for data networks. In comparison to the Internet protocol stack, it includes two extra layers called *presentation layer* and *session layer*. The basic principles behind the OSI design are:

- Create a layer when a different abstraction is needed.
- Implement in each layer a well-defined service.
- Minimize information flow across the interfaces.
- Choose services of the layers by taking into account internationally standardized protocols (i.e. their services).

Because of the last principle, the OSI design and the Internet design bear a lot of similarities. In fact, the five layers of the Internet protocol stack are equivalent to the layers 1, 2, 3, 4 and 7 of the OSI protocol stack.

Presentation Layer: Sometimes the data sent from the source needs to be translated into a different format for the destination to understand. The presentation layer deals with the differences in data representations of the communicating entities in terms of syntax. It translates different character sets of hosts using different operating systems to each other, e.g. convert a text file that is ASCII-coded to its EBCDIC-coded version. Compression and decompression of

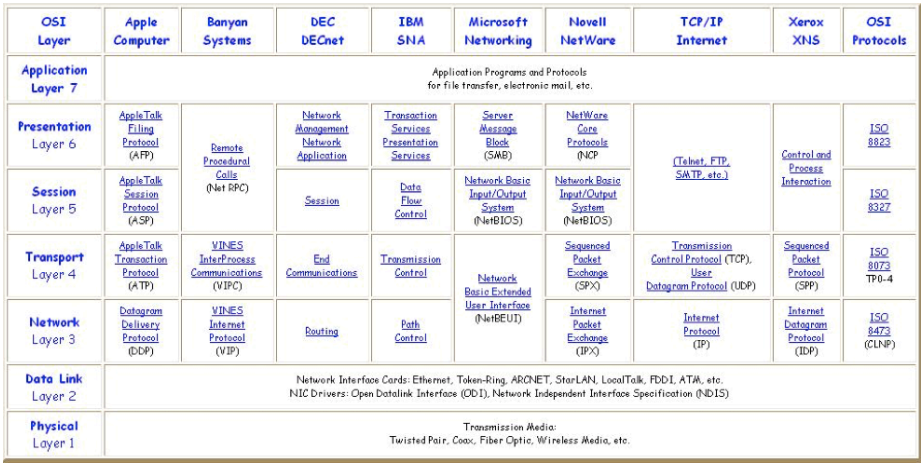


Figure 2.7: Example protocol stacks in relation to the OSI model. (Figure by Radovanovic)

data can be implemented in this layer. Some encryption and decryption protocols can also be implemented, e.g. the Secure Sockets Layer (SSL) protocol. Many network protocol stacks lack this layer since the associated functions are often not needed and they can be implemented in the other layers.

Session Layer: The role of the session layer is to establish, maintain and close communication sessions, i.e. a persistent connection and the corresponding data exchange between two remote processes. An example of a session is a voice call session.

Figure 2.7 gives an overview and examples of different instantiations of the OSI model by different parties.

2.4 Services of a Protocol Layer

Protocol layers serve each other. More specifically, the protocol layer N serves the protocol layer N+1 as shown in Figure 2.8.

> *"A service is a set of primitives (operations) that a layer provides to the layer above it..."*
>
> A. Tanenbaum

Note that services are not the same as protocols. A **service** defines what operations can be performed but it says nothing about how these operations are performed. A **service interface** between different layers defines how to access the services of the lower layer. A **protocol**, on the other hand, determines how exactly the service is implemented and defines a set of rules and packet formats for this purpose.

Data encapsulation is a typical service of a protocol layer. Every protocol layer (except layer 1 which only deals with bits and not data units) implements data encapsulation as one protocol layer's data is payload for the layer below it.

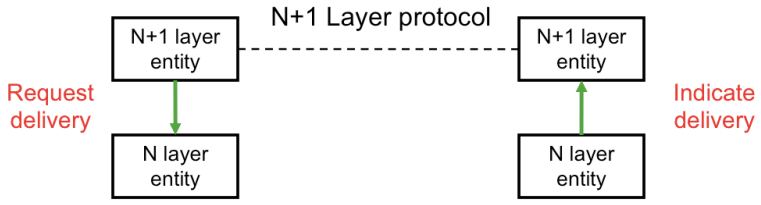


Figure 2.8: Service provided by layer N to layer N+1.

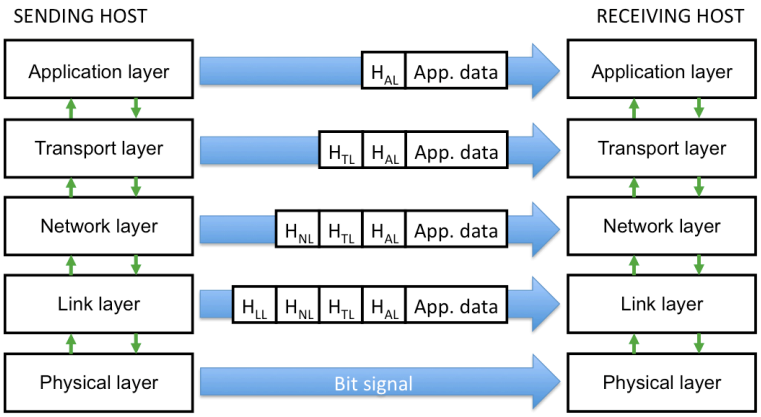


Figure 2.9: Data encapsulation in the Internet.

Every protocol layer deals with its own type of packet, and its own protocols, without depending on the other layers. Figure 2.9 shows data encapsulation in the Internet.

As mentioned earlier, in the Internet terminology:

- A **message** is a layer 5 (application) packet.
- A **segment** is a layer 4 (transport layer) packet.
- A **datagram** is a layer 3 (network layer) packet.
- A **frame** is a layer 2 (link layer) packet.

2.5 Performance: Network Delay, Packet Loss and Throughput

What is the cause of delay? How does the network lose packets? How much data can be communicated over a network every second? Actually, the answers to these questions are somewhat related.

There are four sources of delay in a packet switched network.

Nodal Processing Delay: Routers spend some time doing some simple processing for every packet that they receive in one of their network interfaces. Such processing includes, for example, check for bit errors inside the packet

and determining output link at which the packet needs to be transmitted again. This delay component typically contributes a few microseconds or less to the overall delay.

Queuing Delay: After the nodal processing has been completed, the packet still has to wait in line in a packet queue at the selected outgoing link of the router for being transmitted. This is because packets are transmitted one by one by the routers at each of their outgoing links. The queuing delay depends on the network packet traffic. If there is congestion in the network, then the queuing delay becomes by far the dominating factor in the overall delay.

Transmission Delay: The store and forward behavior means that packets must experience a transmission delay at each router, i.e. time needed for fully transmitting all bits of the packet. This happens at each router (and on each link) that they pass through. The transmission time depends on the **link throughput**[2] of the link and varies across links as we discussed earlier. If the outgoing link throughput is R bps and the packet length is L bits, then the time it takes to send L bits into the link is equal to L/R seconds. Transmission delay can be significant for slow links.

Propagation Delay: It takes time for a single bit (or the signal symbol representing the bit) to take the distance across a wire (or wireless medium). Depending on the transmission medium, the propagation speed of bits is close to the speed of light. However, even at this speed, the propagation delay on very long links (e.g. intercontinental or satellite links) becomes noticeable. This delay (on a single link) is equal to the distance divided by the propagation speed. The propagation delay can vary from a few microseconds to hundreds of milliseconds in practice.

The total nodal delay is equal to the sum of the nodal processing delay, the queuing delay of the node, the transmission delay onto the next link and the propagation delay experienced on that link.

Definition 2.5.0.1 *End-to-End Network Packet Delay*
The time difference from the transmission instance of the first bit of a packet from the source until the reception of the last bit by the destination.

The end-to-end network packet delay is equal to the sum of all four delay components for all nodes passed through before reaching the destination.

Data **packet loss** occurs if a packet arrives to a full queue at a router. In this case, one of the packets in the router's queue (e.g. the newly arriving packet) has to be discarded. This is the reason for losing a lot of packets during network congestion.

If the total delay between the source and the destination is somehow too high, this may also cause situations which effectively translate to data loss. For example, a video frame which passed its playback time during a video session is not useful and cannot be used, i.e. there is no way to fix the past.[3]

[2]The rate (bits per second) at which bits are transferred onto a link.

[3]On the other hand, such data can still be used by some modern video codecs to increase the quality of future video frames.

Throughput gives the rate (bits/second) at which the bits that are pumped into the network by the sender side reach the destination (after some delay, of course). For a good user experience, an application that requires processing of R bits every second (for example, consider streaming of a video that uses R bits to encode every second of footage) needs a network throughput of roughly R bits per second. There is a simple reason for this: the video player needs to consume R bits every second (convert those bits into video frames), which it can only receive through the network. When the network throughput is consistently lower than this minimum requirement R the video at the receiver will inevitably freeze since the receiving device is not getting the bits fast enough to consume them in a timely manner. This is what you experience when your smart phone switches from 4G to 3G while you are watching a high-definition YouTube video.

2.6 Summary

The goal of this chapter was to give the motivation for protocol layering and introduce it as a general network concept. We have explained how this concept applies to the well-known OSI reference model and to the Internet model. You should now be able to study layered protocol stacks and understand the service model provided by each layer. Layering is not optimal performance-wise. However, the gain from reduced complexity and increased flexibility (modular design) is so high that protocol layering is widely used in computer networks. Nevertheless, we also have to acknowledge the existence and the use of vertical protocols in practice as well. A vertical protocol is a single protocol that governs the operation of everything all the way from hardware function to application function. Although this typically results in better performance, it makes maintenance, management and interoperability very tough (e.g. new hardware requires changes to the protocol). We also discussed the concepts of network delay, packet loss and throughput.

2.6.1 Literature

Suggested reading

- Chapter 1 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 1].

2.7 Homework Exercises

1. Consider two family houses with 5 members each, living in different cities and sending each other postcards. The father in each house is responsible for handling the mailbox (picking up mail brought by the postman and placing outgoing mail in the mailbox for the postman to collect). The postman takes the outgoing mail to the local post office, from where it is delivered to the destination post office by other postmen. Finally, the mail is delivered by another postman that works at the destination city to the delivery address. Draw a mail protocol stack and indicate who are in each protocol layer.

2. What does protocol layering mean? How is it useful?

3. Consider the queuing delay in a router buffer with infinite size. Assume that each packet consists of L bits. Let R denote the rate at which packets are pushed out of the queue (bits/sec). Suppose that the packets arrive periodically every L/R seconds. What is the average router queuing delay?

4. How does the average transmission delay affect the average queuing delay? Consider busy routers with a lot of traffic and explain. (Not asking for a mathematical relation.)

5. Make a reasonable discussion of how the different delay types contribute to the end-to-end packet delay in the following cases, i.e. indicate which delay types are likely to have the most influence on each scenario and why.

   - A session between two hosts in Canada and Belgium, outside of peak hours.
   - A session between two hosts in Canada and Belgium, during peak hours (congestion).
   - A session between two hosts in Eindhoven outside of peak hours.

Chapter 3

Application Layer

In this chapter, the fundamental issues addressed by application layer protocols, their service models and their architectural styles are defined. The service models that are needed from the transport layer below are discussed. Finally, examples of well-known application layer protocols are explained. Precisely, we answer the following questions:

- What are the principles of application layer protocols?
- What services do they need from the layer below?
- What are the supporting architectural models?
- What are the details of example application layer protocols?

3.1 Application Layer Protocol: What is it and what is it not?

Let us start by stating that an application layer protocol is not an application by itself! It is a means to easily build protocol-compliant applications on top and provides the services needed for this.

Many network applications have multiple functions that relate to different application layer protocols. An application may leave flexibility to the end user for choosing which implementation of the application layer protocol to use. Sometimes an application (by design) gives the option to modify the protocol handler (or plugin) in the application settings. For example, you can choose in Google Chrome whether you want to use the built-in Flash player of the web browser or the Flash player of the separately installed Adobe Flash Player. Both implement the same application layer protocol, Real Time Messaging Protocol (RTMP) and the application developer (in this case even the application user) is free to choose. Similarly, the same web browser can also use different application layer protocols[1] for streaming different types of videos.

Table 3.1 gives several examples of applications and (some of) the underlying application layer protocols that enable a specific function of the application.

[1] Often there are implementations ready for use, which can be included in the software or implementations that can be used as plugin.

Example applications	Application function	Supporting application layer protocol (example)	Remarks
Web browser (e.g. Chrome, Firefox)	Surf the web: download web pages.	Hypertext Transfer Protocol (HTTP)	HTTP is built into the web browser.
Web browser	Stream Flash video.	Real Time Messaging Protocol (RTMP) defined by Adobe Flash	The browser has a native player that implements RTMP or it uses Adobe Flash Player as a plugin.
Web browser	Stream YouTube video.	HTTP Live Streaming (HLS)	Default browsers in iOS and Android support HLS natively.
E-mail client (e.g. Outlook, Mozilla Thunderbird)	Send and receive e-mails.	Exchange protocol and HTTP for mail in both directions. SMTP for outgoing mail. POP3, IMAP for incoming mail.	Protocols built into the application.
File transfer client (e.g. FileZilla, CyberDuck)	Transfer files from and to a server.	File Transfer Protocol (FTP)	FTP is built into the application client.

Table 3.1: Some applications, the application layer protocols supporting their functions. The list of application protocols is not meant to be complete or comprehensive. It serves as an example.

In reality the number of applications that can be created is unlimited. Figure 3.1 shows the increase of the number of apps available in the iOS App Store over the years. Not all of these apps are network apps, of course. However, a significant percentage are network apps, i.e. their usefulness without a network connection is very limited.

Many times you are asked to give privacy permissions to these apps, e.g. access to your contact list and the corresponding use policy, which could be very sensitive. Apple has a strong security policy (probably the best enforced in this market) that only the apps that are available from the official iOS App Store can be installed on iOS devices. They check the security settings of each application from many developers (and the overall compliance to Apple security and privacy guidelines) and those apps that violate these policies are rejected (i.e. if they are detected). Applications that do not pass Apple's review process can still be found available for download in various web pages. However, you will not be able to install them on an official 'original' iOS release. As we discussed in Chapter 1, there is still the question of "what is security?" here. The attacker model does not contain the errors made during the screening process. For example, news reports indicate that in 2015 around 250 apps were withdrawn by Apple due to a privacy policy breach[2]. There were a total of about one million downloads of these apps "capable of collecting personal data" before the issue was finally discovered.

The communicating entities in an application layer protocol are **not** users

[2] `http://www.itv.com/news/2015-10-20/apple-withdraws-hundreds-of-apps-after-privacy-breach/`

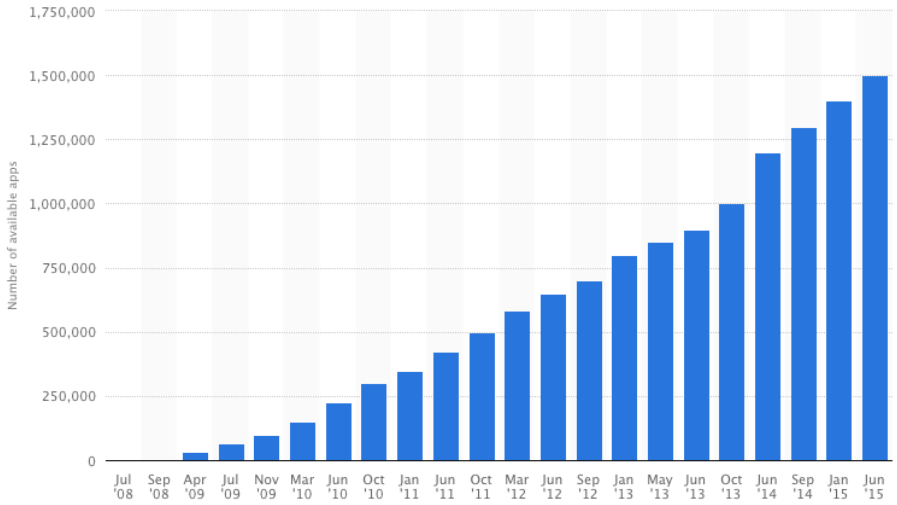


Figure 3.1: The number of apps in the Apple app store over years. (Figure source: statista.com)

or end devices. They are application programs (or rather processes), which are most of the time remotely located. These may also reside on the same end device, e.g. two processes running in the same computer may communicate over a network connection.

Furthermore, application layer protocols are not for the network core. A design goal, which is particularly critical for large networks like the Internet, is to push complexity to the network edge. In the Internet, a router does not run user applications and application layer protocols because it would be just too complex considering the combination with all the packet traffic and processing it has to deal with. However, it is not strictly forbidden or impossible to put an application in places other than end devices, e.g. for debugging or testing purposes. In this course, we will disregard these exceptions and consider that routers always implement up to layer 3 (network layer), and that layers 4 and 5 (transport and application layers) are implemented only by end devices. Computer network applications of users live on **end devices** that communicate over a network infrastructure as shown in Figure 3.2. An example is the relation between a web browser program and a web server program.

3.2 Issues Solved by the Application Layer

Generally speaking, the issues that are addressed by various application layer protocols are:

1. **Architecture Styles:** An application protocol can utilize the pure client-server architecture style or the pure peer-to-peer architecture style. Alternatively, it can also utilize a hybrid of the two architecture styles.

2. **Addressing:** Remote entities of a network application communicate with each other using addresses. Each application provides an address format

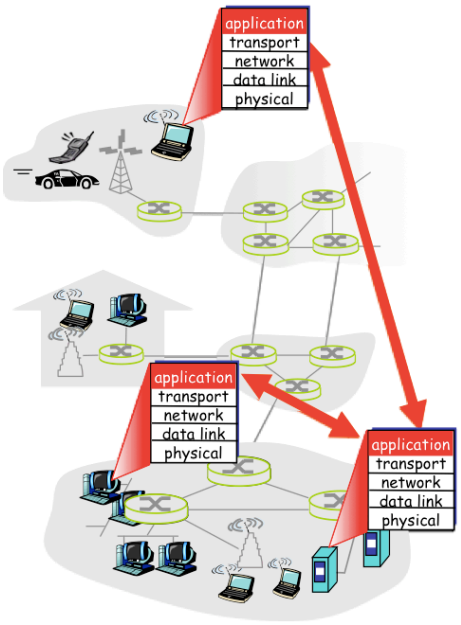


Figure 3.2: Network applications and their protocols live on end devices. (Figure by Kurose and Ross)

for this purpose. An example e-mail address is `student@tue.nl`. An example web address is `http://www.tue.nl` or, more precisely, `http://www.tue.nl:80` where 80 denotes the default port number for HTTP. The web client process finds the web server process using its "socket address" (covered in detail in Chapter 4) consisting of its IP address and port number. The translation from a domain name (e.g. `tue.nl`) to an IP address is done by the application layer's Domain Name System (DNS) protocol.

3. **Service Models:** The application layer protocol provides services to the applications running on top, and receives services from the transport layer. For example, the Simple Mail Transfer Protocol (SMTP) is a service for the mail client applications that allows the application to send e-mail messages to an e-mail server. Also see the examples in Table 3.1.

3.2.1 Application Architecture Styles

In all three models explained below the communicating entities are remote application programs; i.e. they are **not** users or devices.

The Client-Server Model

For a long time the client-server architecture model was "the" model for Internet applications. Below are definitions of *client process* and *server process*, where the term **process** refers to a "program under execution". A program is just

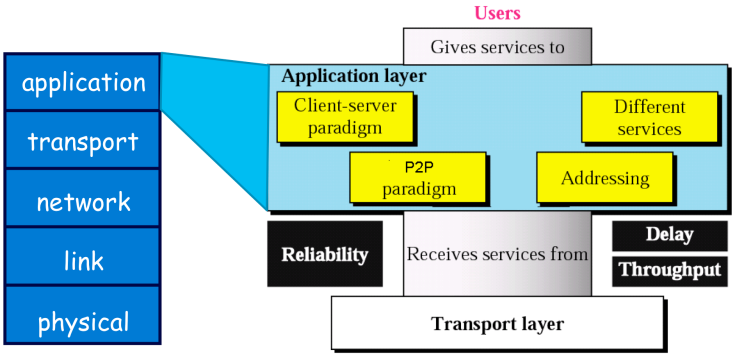


Figure 3.3: Issues solved by application layer protocols. (Figure by Forouzan)

program code defining variables and actions on variables. The operating system can create multiple processes from the same program. For example, you can have multiple instances of a web browser program execute concurrently.

Definition 3.2.1.1 *Client and Server Processes*

In the client-server architecture, the process that initiates the contact ("speaks first") and requests services from a server is called a client process. The server process has a fixed entry point for incoming requests, i.e. the 'location' (network address and process identifier) of the server needs to be fixed.

In this model, there is an always-on server (or a set of servers that share the load) that runs an infinite server program at all times, 24/7, always ready to serve requests of client processes. The server typically has access to an ample amount of computation, storage and bandwidth resources, and has a permanent network address (e.g. an IP address in the Internet). The client on the other hand may be online or offline (on demand e.g. of its user) and can change its (IP) address.

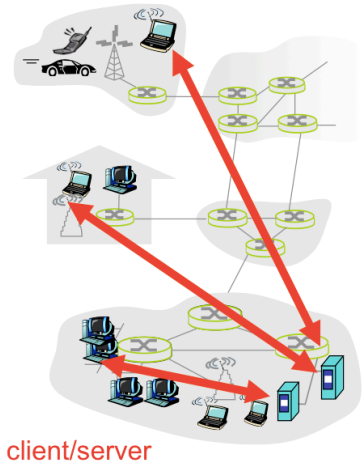


Figure 3.4: Client-server model. (Figure by Kurose and Ross)

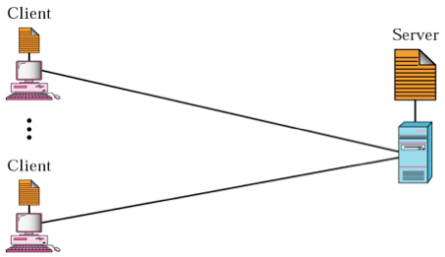


Figure 3.5: Multiple clients (client processes) connecting to a single server (process) in the client-server model. (Figure by Forouzan)

Typically a lot of clients are served by a server. As of April 2016, there are 7000+ Tweet's posted, 700+ Instagram photos uploaded, 53000+ Google search queries submitted **every second**[3]. This translates to huge client-server traffic. In order to deal with these scalability issues service providers build very large server farms that by themselves need huge amounts of power (Watts). That is why server farm locations have to be chosen very carefully looking at climate and nature (e.g. for water cooling).

Pure clients do not communicate directly with each other and if they do, at least one of them has to have **both** client and server parts. If both have client and server parts but they do not exactly act like a server (e.g. not always-on) then the difference from the peer-to-peer model becomes a bit blurry. Note that even applications with peer-to-peer architectures have client processes and server processes (i.e. one initiates the connection and the other accepts it).

Peer-to-Peer and Hybrid Applications

In the peer-to-peer architecture style there is no always-on server process and peers that arbitrarily communicate with each other can go online and offline as they wish. Peers can also change their IP addresses.

Unlike the client-server relation, peers do not have a reference IP address (of another peer) where they can initiate the communication. In order to be able to communicate, they first need to deal with this problem by making use of a list of reference entry points (e.g. a list of IP addresses that are potentially online). For example, in the Gnutella peer-to-peer network (e.g. LimeWire) a newly joining peer repeatedly tries a list of candidate peers until it gets a (TCP) connection to a remote peer. The new peer then sends a (ping) message to this remote peer, which is forwarded (flooded) to a number of other peers. In return, these peers reply with a (pong) message, allowing the new peer to make connections to them.

File sharing applications, e.g. those based on the BitTorrent protocol, are very popular. This protocol is based on pure peer-to-peer connections for file transfer and client-server connections for peer tracking. Servers that are called "trackers" host the list that contains references to the files to be shared.

[3] `http://www.internetlivestats.com/one-second/`

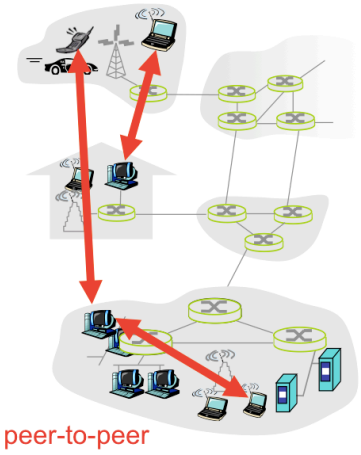


Figure 3.6: Peer-to-peer model. (Figure by Kurose and Ross)

3.2.2 Addressing in Application Layer

Application addressing concerns the ability of an application to uniquely refer to relevant network resources (e.g. resources like a web page or an image). The combination of an IP address (e.g. 131.155.74.72), with a port number (e.g. 80) and a directory structure (e.g. `/~tozceleb/2IC60/index.html`) allows us to address any resource in the Internet.

Just try `http://131.155.11.13:80/~tozceleb/index.htm`.

However, this type of addressing is not very convenient to use in an application. For this reason, the Internet protocol stack defines identifier formats that are universally applicable and easier to understand. These identifiers are called Uniform Resource Identifiers (URI). A specific type of URI is called an Internet address or a Uniform Resource Locator (URL) and it points to the location of the resource and the way to reach it. The URL structure is depicted in Figure 3.7.

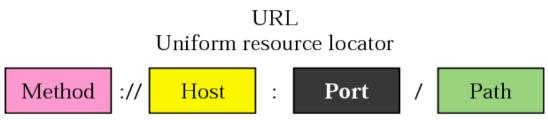


Figure 3.7: URL structure. (Figure by Forouzan)

URL fields are

- Protocol: indicates the protocol (e.g. FTP, HTTP).
- Host: a computer where the resource is located.
- Port: the (default) port number of the destination process.
- Path: the path name of the file (the resource).

The URL `http://www.win.tue.nl/~tozceleb/2IC60/index.htm` for the course website is much easier to remember than the IP address and the path name given above. Or equivalently: `http://www.win.tue.nl/~tozceleb/2IC60/`

Including the method (`http`) the URL consists of four parts. The second part is the host name `www.win.tue.nl`. The third part is the port number (80). Finally, the fourth part is the path name `~tozceleb/2IC60/index.htm`.

In practice the port number 80 and the object name "index.htm" are assumed by modern web browsers and need not be explicitly stated.

Definition 3.2.2.1 *Uniform Resource Identifier (URI)*
"A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource." (IETF RFC 2396)

As we will see later in Chapter 5 routing in the Internet is done using IP addresses, so there has to be a mechanism in place that translates a *name* into an *IP address*. The role of the application layer protocol Domain Name System (DNS) is exactly this. We will go into the details of DNS in Section 3.5.2.

3.2.3 Application Layer Services

The application layer protocol defines types of messages exchanged between communicating processes (e.g. request, response etc.), the message syntax (i.e. the message fields and formats), message semantics (i.e. meaning of information in different fields) and rules for message exchange (i.e. when and how processes send and respond to messages). Protocol details can be publicly available (RFC 2616 for HTTP and RFC 2821 for SMTP) or proprietary (e.g. Skype).

3.3 Layer 4 Services Used by an Application

In providing these services for the applications, the application layer makes use of the services of the transport layer. The categories of services that may be received from the transport layer **in theory** are *reliability*, *delay* and *throughput* support as depicted in Figure 3.3.

- **Reliability support:** Applications such as e-mail and file transfer are intolerant to data loss . In this case, the transport layer's reliability service (e.g. TCP covered in Chapter 4) can be used by the application to make sure that no data is lost in transit.

- **Delay/Timing support:** Some applications may have constraints of maximum delay or maximum delay jitter (variance of delay), e.g. real-time video conferencing. The application protocol may rely on the services of the transport layer protocol to mitigate the effects of delay and delay jitter.

- **Throughput support:** The number of bits transferred per second is important for many network applications (e.g. multimedia) for quality purposes. Other apps (e.g. file transfer) make use of what throughput they get.

- **Security support:** The transport layer can provide a security service. Example services are confidentiality (through encryption) and data integrity (through hashing).

Some examples of transport layer service requirements of different applications are given in Figure 3.8.

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-10Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Figure 3.8: Transport service requirements of some applications. (Figure by Kurose and Ross)

A transport layer protocol may provide some of these services. For example the Transmission Control Protocol (TCP) of the Internet's transport layer is connection-oriented: a connection setup is required between client and server processes. It provides reliable transport of application messages between sending and receiving processes and it makes sure that packets are delivered to processes in the correct order. It provides flow control between the sender and the receiver such that the sender does not overwhelm the receiver by sending too much, too fast. It is sensitive to increasing network traffic and implements congestion control by throttling the sender process when network delays are high (i.e. long router queues). However, TCP as a protocol does not provide delay, minimum throughput or security guarantees.

User Datagram Protocol (UDP), on the other hand, does the bare minimum expected from a transport layer protocol, that is process to process delivery of data. UDP does not provide connection setup, reliability, flow control, congestion control, timing guarantees, throughput guarantees, or security.

Why do we bother with UDP when TCP seems to provide all those services that UDP does not? The answer lies in the simplicity and the speed of UDP. Some services that are provided by TCP may not be needed. For example, if the link is already reliable, you don't need additional reliability from the transport layer protocol. Moreover, if for some reason the application runs on top of UDP, those extra services that are needed can always be implemented in the application layer protocol (e.g. reliability). We will come back to UDP and TCP later in Chapter 4.

Examples of application layer protocols and their underlying transport layer support are given in Figure 3.9.

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g. Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Figure 3.9: Transport layer protocols supporting various applications and application layer protocols. (Figure by Kurose and Ross)

3.4 Programming Network Applications

Making network applications requires the ability to call the functions that enable process-to-process communication, which is a service originally provided by the transport layer. For simplicity, these services need to be available to the application developer as part of an Application Programming Interface (API). Using the **socket API** a network application can create and access network sockets (i.e. send and receive messages).

Definition 3.4.0.1 *Network Socket*
A network socket is a data structure that represents one end of an end-to-end (process-to-process) connection. A process talks to the outside world through network sockets.

In order to address the destination process at the other side of the connection, a process needs an identifier for the process, or equivalently, an identifier for the corresponding socket.[4]

In the Internet, processes are identified by using IP addresses and port numbers. For example, a web browser addresses a web server using the IP address of the web server (which is resolved from a web address) and the default HTTP server port number 80. We will cover network sockets and process-to-process addressing in much more detail in Chapter 4.

3.5 Example Application Layer Protocols

3.5.1 Hypertext Transfer Protocol (HTTP)

The world-wide web was introduced in 1991 as an Internet application. HTTP is a client-server based application layer communication protocol that governs

[4]This is not to mean that each process has one socket. Just like a house can have multiple doors, a process can create multiple network sockets.

the world-wide web. Before we explain HTTP services, let us define some web terms that will be used in this discussion. A web page consists of objects such as HTML (Hypertext Markup Language) files, images, Java applets, and audio files. Each object is associated with a URL. The HTML file is the base file that contains text and several references to the other objects of the web page in the form of URLs.

The client process (e.g. a web browser like Google Chrome, Internet Explorer) requests, receives, and displays web objects. The server process (e.g. an Apache web server) sends objects in response to HTTP requests as shown in Figure 3.10. HTTP uses a TCP connection on the well-known HTTP port number 80. In one of the lab sessions you are asked to build such a web server.

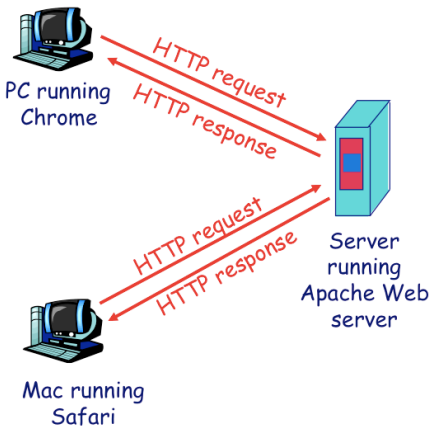


Figure 3.10: HTTP request and response. (Figure by Kurose and Ross)

Figure 3.11 shows the message sequence diagram created as a result of a user entering the URL `www.win.tue.nl/index.html` in her web browser. The example shows the operation with a non-persistent connection, i.e. the web server closes the TCP connection after each response and the client needs to open a new TCP connection for every request. In case of a persistent connection (default in HTTP v1.1), the server keeps the TCP connection open for further requests of the client. Using persistent connection the client can pipeline its requests without waiting for individual responses. Actually, also with non-persistent connections, the client can open parallel TCP connections to pipeline its requests. However, this is less efficient in terms of message traffic.

There are two types of HTTP messages: request (from client to server) and response (from server to client). HTTP messages are in ASCII (human readable text) format.

HTTP request message: The exact format of the HTTP request message is as shown in Figure 3.12. An example HTTP request message is given in Figure 3.13.

- **Request line:** This is the first line that indicates the type of the request (method), the directory "path" of the requested object URL, and the HTTP version.

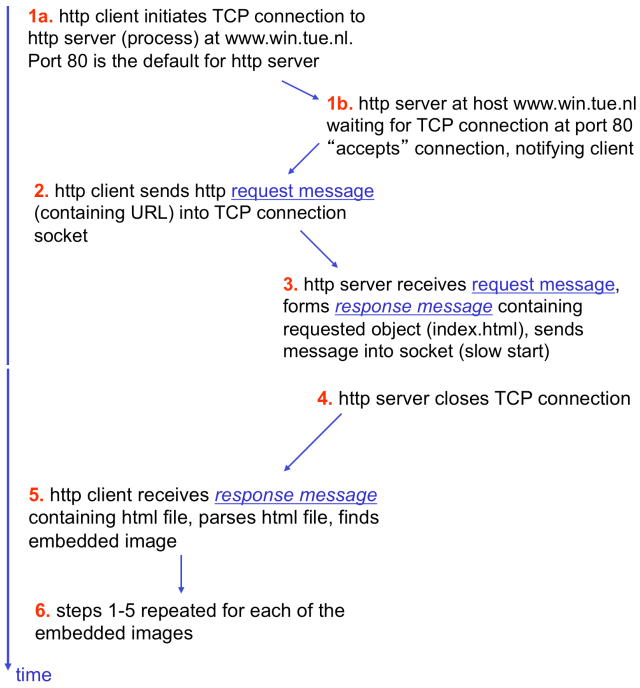


Figure 3.11: HTTP message flow diagram upon entering the URL `www.win.tue.nl/index.html`. (Figure by Kurose and Ross)

- **Header lines:** The header lines in this field contain information such as the "host" part of the requested object's URL, the name of the web browser, the requested type of connection (persistent or not) and the requested language (useful if the web server provides versions in multiple languages, e.g. when you want to view the web page in Dutch).

- **Carriage return, line feed:** Indicates the end of the HTTP request header. The body of the message (i.e. the application payload) follows this.

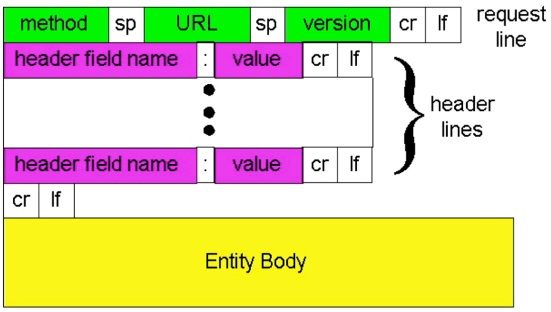


Figure 3.12: HTTP request message format. (Figure by Forouzan)

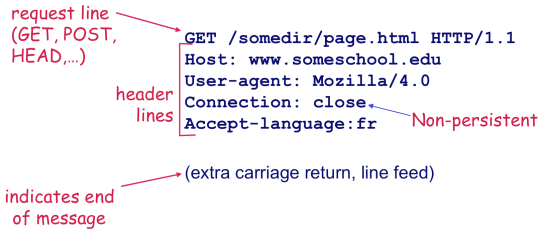


Figure 3.13: HTTP request message example. (Figure by Kurose and Ross)

HTTP response message: The exact format of the HTTP response message is as shown in Figure 3.14. An example HTTP response message is given in Figure 3.15.

- **Response line:** This is the first line that contains the HTTP version and the response code, e.g. **200 OK**, **404 Not Found**.
- **Header lines:** The header lines in this field contain information such as type of connection (persistent or not), date and time of response, server software identification, date and time of last modification for the object included in the response, the object length and the object type.
- **Carriage return, line feed:** Indicates the end of the HTTP response header. The requested object follows this.

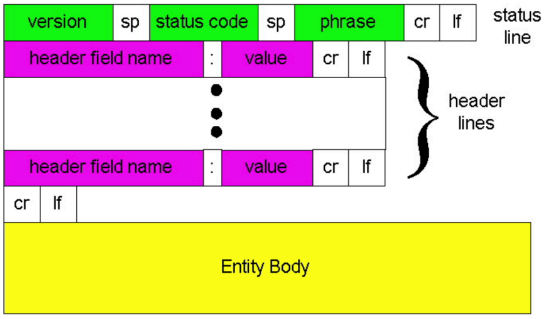


Figure 3.14: HTTP response message format. (Figure by Forouzan)

Cookies: How to Keep State

In its original design, HTTP is a stateless protocol, i.e. it defines request and response transactions without knowledge of and regardless of the current states of the client and the server. That is the server does not store contextual information of a client such as: what was the last object that was downloaded? This feature contributes to the simplicity of the protocol on the one hand, and the privacy of the clients on the other.

However, state keeping has its advantages too. For example, your online book store can maintain a list of books you previously looked into and recommend similar books. For this purpose, cookies are provided as an additional tool

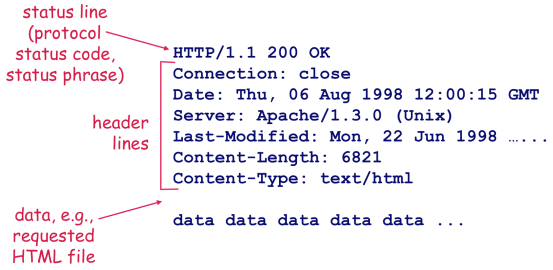


Figure 3.15: HTTP response message example. (Figure by Kurose and Ross)

to keep state and HTTP's cookie header line (added after cookie introduction) facilitates this.

A cookie is a file created and stored by the client based on an HTTP request-response interaction and it is managed by the web browser. Every time the client visits the same website the cookie identity is communicated inside the HTTP request message. The web server can do some cookie specific action with this information (e.g. book recommendation by the bookstore) using the data accumulated about this specific client at a back-end database. This is illustrated in Figure 3.16.

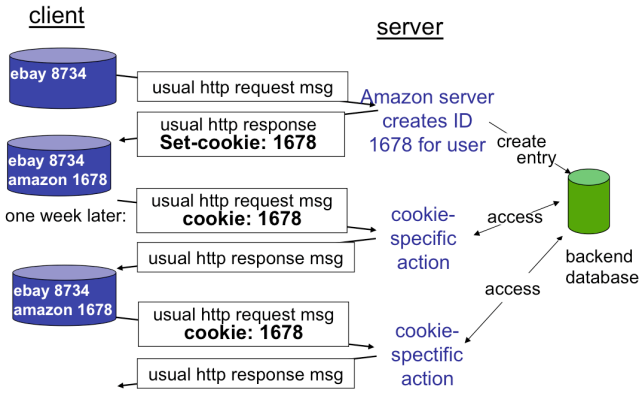


Figure 3.16: Cookie use. (Figure by Kurose and Ross)

Since cookies allow a website to gather information about a user (what you like, what you search etc.) it can potentially allow third parties to identify you, which is a clear **privacy** issue.

Web Cache / Proxy Server

HTTP supports proxy servers, or as they are otherwise known, web caches.

Definition 3.5.1.1 *Proxy Server / Web Cache*

A proxy server or a web cache is a network host that can respond to HTTP requests on behalf of a web server, i.e. without involving the web server in the request-response exchange.

The proxy server achieves this by keeping copies of HTTP responses to recent requests. In order to enable this feature, the user needs to configure her web browser to access the Internet via the proxy (i.e. proxy settings). After that the web browser sends its HTTP requests directly to the proxy server. If the object requested is not in the proxy, the proxy sends a request to the origin server for the object. After receiving the object, the proxy forwards it to the client in an HTTP response message. However, if the requested object is already at the proxy, the proxy sends the object back to the client immediately, without contacting the origin web server. This is shown in Figure 3.17. This can significantly reduce the average response times for HTTP requests and reduce traffic on the access link to the rest of the Internet.

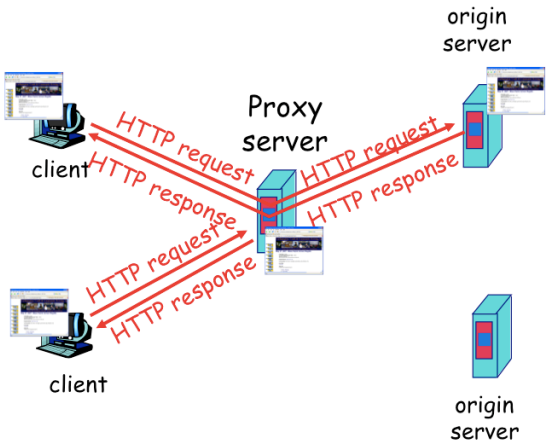


Figure 3.17: Web cache operation. (Figure by Kurose and Ross)

Web caches can also provide an important security advantage by scanning the objects they receive from the origin server. Behind a web cache the client computer is less likely to be infected by malware and viruses.

3.5.2 Domain Name System (DNS)

The devices in the Internet are identified in different ways. For example, a web server has a unique domain name (e.g. `tue.nl`) and a unique IP address. Domain Name System (DNS) is the application-layer protocol of the Internet that utilizes distributed database (many servers) and is responsible for dealing with:

- Address resolution.
  - Translating a domain name into an IP address and vice versa.
  - Inverse translation is also useful, e.g. when a server needs to verify the country of origin of a client.
- Host aliasing, i.e. maintaining canonical and alias names.

  - E.g. `www.gmail.com` versus `mail.google.com`.

- Replica web servers: a set of IP addresses that map to a single canonical name.

  - E.g. mapping to various server IP addresses for the domain name `www.google.com`.

In the hierarchical name space of DNS, each name is made of several parts. This hierarchy corresponds to a tree structure as shown in Figure 3.18. A domain is nothing but a subtree of this tree structure. Each domain is identified by a domain label and the top level domain labels are, e.g. generic (3 letters) such as com, edu, org, net, or per country (2 letters) such as nl, fr. Finally, a domain name is a sequence of labels separated by dots, e.g. `win.tue.nl` is with labels `win`, `tue` and `nl`.

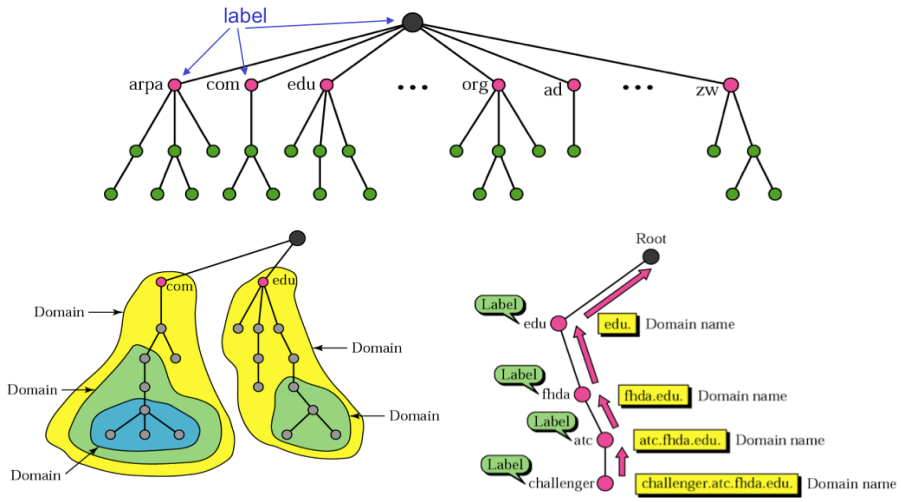


Figure 3.18: DNS name space tree structure. (Figure by Forouzan)

If a DNS server knows how to translate the queried domain name, it responds with the translation. If a DNS server does not know how to translate a particular domain name, it asks another one, and so on in a hierarchical manner, until the correct IP address (or domain name in case of inverse translation) is returned. Two types of DNS query handling are shown in Figure 3.19, where a host at `cis.poly.edu` wants to find out the IP address for `gaia.cs.umass.edu`. On the left, a recursive DNS query to the authoritative DNS server is followed by a number of iterative DNS queries. On the right, the figure shows all recursive DNS queries. In the "all recursive" case the requesting host puts the burden of name resolution on the hierarchy of DNS name servers, which results in heavy load for the DNS infrastructure, especially when we consider huge numbers of queries (what happens in reality). The use of both UDP and TCP is possible in DNS, on port number 53.

The DNS server hierarchy is exemplified in Figure 3.20, where the Top-Level Domain (TLD) servers are responsible for all top-level country domains as well

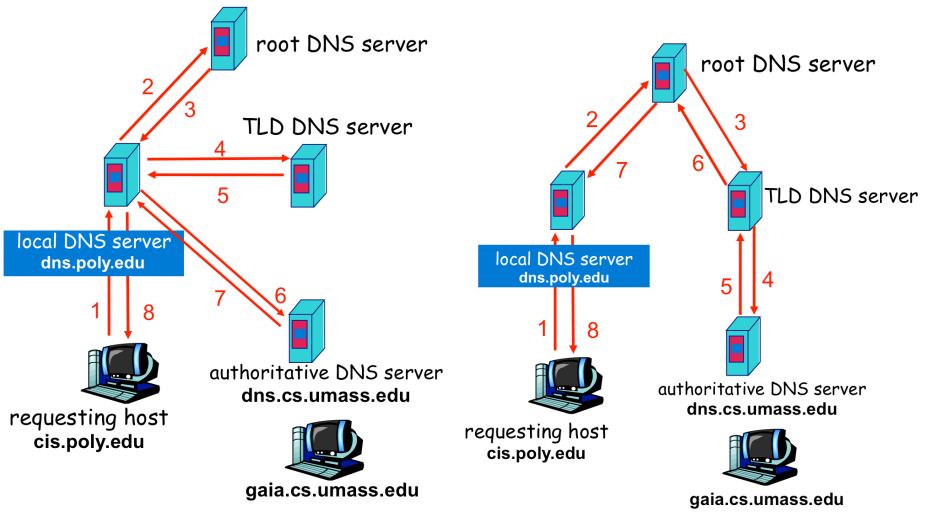


Figure 3.19: DNS queries. Left side: iterative and recursive queries mixed. Right side: All recursive queries. (Figure by Kurose and Ross)

as for com, org, net, edu, etc. Authoritative DNS servers are DNS servers of individual organizations. These DNS servers provide authoritative hostname to IP mappings for an organization's servers (e.g. Web, mail). They can be maintained by the organization or its service provider.

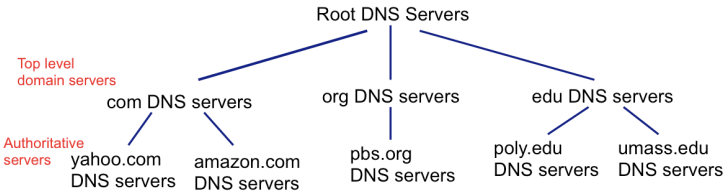


Figure 3.20: Distributed, hierarchical DNS database. (Figure by Kurose and Ross)

A final remark on DNS: Even though it is an application layer protocol, it provides a fundamental Internet function. For example, without DNS, web surfing would not be practical (people would have to remember IP addresses instead of web addresses).

3.5.3 BitTorrent

In peer-to-peer file sharing, a peer starts to serve as a file source as soon as it receives some part of the file from another peer, i.e. it shares what it has already received. Due to difficulties in managing content that is shared, peer-to-peer file sharing has been subject to many lawsuits.

BitTorrent is an application layer protocol for file sharing. Here the term **torrent** refers to a set of peers sharing parts of a file (i.e. each peer shares the

parts it has).

BitTorrent with Trackers: The original BitTorrent is a hybrid protocol that is a combination of peer-to-peer file sharing with a client-server architecture based peer tracking scheme. Servers that are called "trackers" host the list of files to be shared (but not the files themselves) and they "track" which peers are in possession of file parts. By connecting to a tracker, the client program can find peers and establish TCP connections with them.

Tracker-less BitTorrent: Recently "tracker-less" BitTorrent protocol gained some momentum, where tracker servers are no more needed and their peer lookup services are replaced by Distributed Hash Tables (DHT).

Exchanging File Chunks: In BitTorrent the shared files are first divided into chunks of size 256 KB. When trackers are used a new peer registers with the tracker to get a list of peers that are participating in the same torrent. This new peer does not yet have any chunks and starts accumulating chunks after making TCP connections to some peers. The peers with which a TCP connection is established are called **neighbors**. The new peer starts downloading the chunks it does not have. For that it asks each of its neighbors for a list of chunks that they have and it requests the chunks it does not yet have. In doing so it starts with the most rare chunks. In the meantime it uploads the chunks it has already accumulated to other peers. Every 10 seconds a peer ranks its neighbors based on the respective download rates from these neighbors. The peer sends chunks to the top four neighbors in this ranking.

Every 30 seconds a peer (say Alice) randomly selects another peer to send chunks to so that new peers (who do not have many chunks to share) get a chance to make it to the top four list of other peers and ramp up their download speeds. If the new peer (say Bob) reciprocates, it may get into the top four senders list of Alice, effectively ramping up his download and upload speeds.

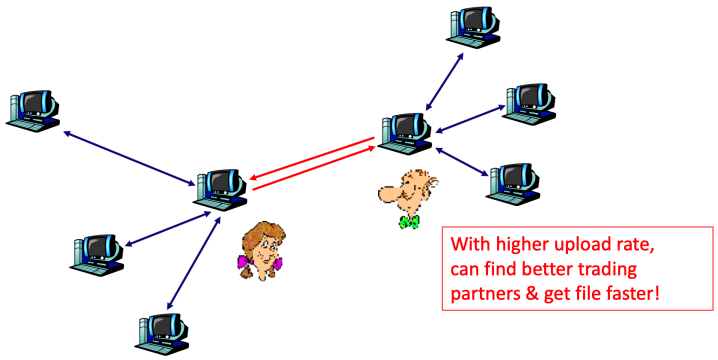


Figure 3.21: The list of top 4 providers for a peer changes dynamically, over time leading to larger cumulative upload and download speeds . (Figure by Kurose and Ross)

After receiving the entire file the peer continues to upload without downloading.

File Distribution Time: In the client-server paradigm, the time it takes to distribute a file to N clients from one server is highly dependent on the file size F, the number of clients, the upload bitrate of the server u_s and the individual download bitrates of the clients (d_i for client i). This delay is necessarily larger than the time it takes for the server to upload N copies of the file (i.e. NF/u_s) and it is necessarily larger than the time it takes for the slowest client to download F bits (i.e. $F/min(d_i)$).

On the other hand, in P2P file sharing, the distribution time of a file is necessarily larger than the time it takes for the first copy to be uploaded by the source peer (i.e. F/u_s). It is larger than the time it takes for the slowest client to download F bits (i.e. $F/min(d_i)$). Finally, it is also necessarily larger than the time it takes for N copies of the file to be uploaded by all peers together (i.e. $NF/(u_s + u_1 + u_2 + \cdots + u_N)$).

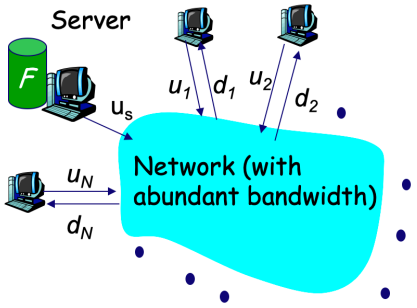


Figure 3.22: P2P file distribution. (Figure by Kurose and Ross)

3.5.4 Blockchain

Blockchain is a protocol for building applications for distributed bookkeeping purposes. Consider a bank that is responsible for bookkeeping of transactions of the bank clients. The bank transaction records represent a centralized ledger that contains such bookkeeping while hiding transactions of individual clients from each other and from other parties. Alternative to this is a decentralized ledger implemented on a distributed P2P network, i.e. a blockchain. The ledger is maintained by the peers and it is important to make sure that the transactions are consistent from the perspectives of all joining peers. A ledger consists of a chain of records (data structures) that are called "blocks", hence the name "blockchain". $Block_i$ contains a record including some transactions, a timestamp and an encrypted hash of the previous block, $H(Block_{i-1})$ as shown in Figure 3.23.

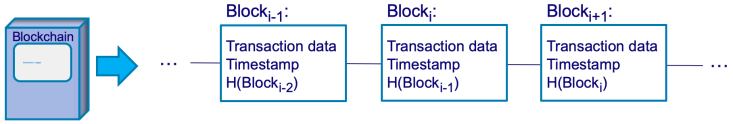


Figure 3.23: Blockchain illustrated.

In a blockchain every peer can see and verify all transactions. Note that this is the opposite of traditional bookkeeping practice. Inserted blocks are there to stay, i.e. it is only possible to add information to a blockchain (no deletions).

3.6 Summary

The goal of this chapter was to provide an overview of the application layer, including possible architectural styles and its service models as well as the services it needs from the transport layer. The socket API gives the necessary tools to program network applications. The concepts introduced were exemplified by three famous application layer protocols. The reader should now be able to make the separation between an application and the application layer protocol that supports the application.

3.6.1 Literature

Suggested reading

- Chapter 2 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 2].

3.7 Homework Exercises

1. Look up the Secure Sockets Layer (SSL) protocol (and its successor TLS) on the Internet. It is an enhancement on top of TCP for secure process-to-process communication over a network. Which layer of the Internet protocol stack do these protocols belong to and WHY?

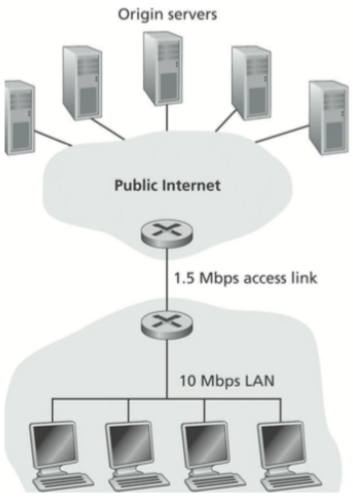
2. What are the transport service requirements of your favorite online game? If you are not a gamer (shocking) or the requirements for the game are vague (e.g. it requires 'broadband connection') then do this exercise for another Internet application of your choice.

3. Simple Mail Transfer Protocol (SMTP) is a protocol that is used to transfer e-mail messages i) from the web client of a sender to his e-mail server, and ii) between e-mail servers of the sender and the receiver(s). Just like in HTTP messages, in an e-mail message, the body of the message is preceded by the SMTP header (carriage return, line feed in between) that consists of plain text. SMTP header fields contain useful information, e.g. for system administrators to trace e-mails. Reading a header from bottom to top, the 'Received:' header lines show a list of all the network entities that the message passed through. View and copy the source of an e-mail message that you have received (you may simply create a test message from your personal e-mail to TU/e e-mail). Answer the following questions.

   - How many 'Received:' header lines are there?
   - What servers and computers did the message go through?

4. List two advantages of deploying a Web cache (i.e. a proxy server) in an institutional network. Justify your answer.

5. The figure shows an institutional network connected to the Internet.



Suppose that the average object size is 900,000 bits and that the average request rate from the institution's browsers to the origin servers is 1.5 requests per second. Also suppose that the amount of time it takes from when the router on the Internet side of the access link forwards an HTTP request until it receives the response is two seconds on average. In this exercise we model the total average response time as the sum of the average access delay (that is, the delay from Internet router to institution router) and the average Internet delay. For the average access delay, use $\triangle/(1 - \triangle\beta)$, where $\triangle$ is the average time required to send an object over the access link and β is the arrival rate of objects to the access link.

(a) Find the total average response time.

(b) Now suppose a cache is installed in the institutional LAN. Suppose the hit rate is 0.4. Find the total average response time.

6. The DNS service in the Internet is distributed by design. Alternatively, DNS could have a centralized design instead. List two disadvantages of a centralized design. Justify your answer.

Chapter 4

Transport Layer

The main responsibility of the transport layer is process-to-process data delivery. Some other requirements of example applications from the transport layer are listed in Figure 4.1.

Application/ requirement	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

Figure 4.1: Comparison of applications with respect to their reliability, delay, delay jitter and bandwidth (throughput) requirements. Rating 'high' in a column means that the application performance is very sensitive to this attribute.

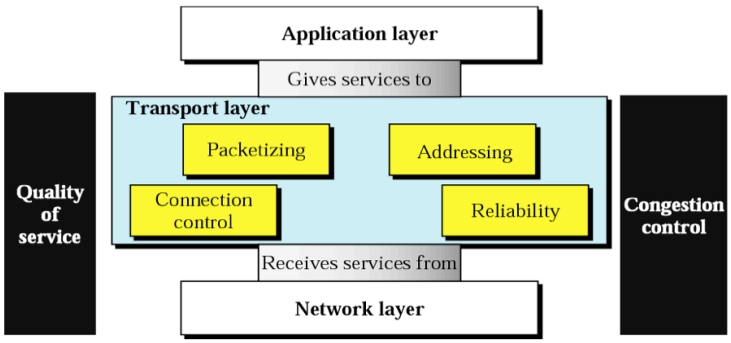


Figure 4.2: The position of the transport layer in the Internet protocol stack. (Figure by Forouzan)

In this chapter, the issues addressed by transport layer protocols and trans-

port layer service models are introduced. The two transport layer protocols of the Internet, UDP and TCP are explained and standard problems are discussed in this context. Precisely, we try to answer the following questions:

- What are the principles of transport layer services and protocols?
- How to realize reliable process-to-process delivery, flow control and congestion control?
- What are the details of transport layer protocols of the Internet?

4.1 Issues Solved by the Transport Layer

Generally speaking, the issues that are addressed by various transport layer protocols are:

1. **Addressing and Packetization:** The addressing taken care of the transport layer works with IP addresses (not with URLs). Recall that the translation from human readable URLs to IP addresses is the job of the application layer and is taken care of by the DNS protocol as we discussed in Chapter 3 (i.e., an example of a fundamental Internet service realized by an application layer protocol). A sender process addresses a remote process using its socket address, whose components are IP address(es) and port number(s). The sender process shoves a message "through the socket", which is then encapsulated in transport layer segments (multiplexing). At the receiver side the message is extracted from the segment and is passed to the correct destination process (demultiplexing).
2. **Connection(-less) Service:** A transport layer protocol may provide connection service, which makes sure that the packets arrive at the receiving process in the same order that they were sent. For this it has the mechanisms in place for buffering and reordering segments. The communicating processes need to establish, maintain and tear down connections in an orderly fashion. The alternative to this is a connectionless service, which aims to deliver messages to the destination individually and in any order that the underlying network accommodates.
3. **Reliable Data Transfer:** Sender processes may rely on the transport layer protocol to reliably transfer their messages to remote receiver processes. Reliability at the transport layer is necessarily through end-to-end (process-to-process) retransmissions. This may be combined with link layer reliability (discussed in Chapter 6) where retransmissions are on a single link. An interesting question (that we will answer later) is the following. Why would we need link layer reliability when we already have transport layer reliability, and vice versa?[1]
4. **Flow Control and Congestion Control:** Flow control aims to prevent a sending process from overwhelming a receiving process by sending too much too fast. Congestion control, on the other hand, aims to prevent sender processes from overloading the network core.

[1] See the lecture slides for an explanation of the generic principles of reliable data transfer. In this text we will focus on the reliable transfer service provided in the Internet's transport layer.

4.2 Addressing Processes and Packetization

For process-to-process data delivery the transport layer needs the host-to-host delivery service of the network layer. Similarly, the network layer builds its host-to-host data delivery service upon the node-to-node data delivery (on a single link) service of the data link layer. This is depicted in Figure 4.3.

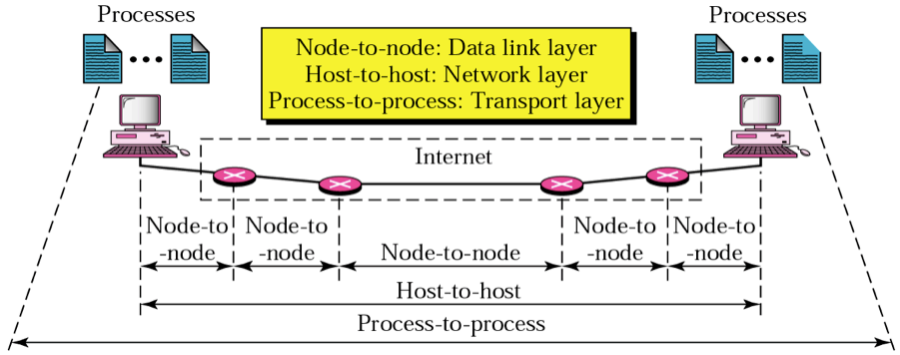


Figure 4.3: Connections at the transport layer (process-to-process), the network layer (host-to-host) and the link layer (node-to-node) shown in one picture. (Figure by Forouzan)

Transport Layer Multiplexing: At the sender side, the transport layer receives data from sender processes and encapsulates these in transport layer segments by adding transport layer headers to them. Finally, these segments are passed to the network layer for further processing.

Transport Layer Demultiplexing: At the receiver side, the segments that are passed from the network layer to the transport layer are examined and the data extracted from these are directed to their respective destination processes.

The destination IP is enough to address the destination host. However, by itself, it is not enough to point to the destination process. Individual processes running on a given host are identified by their port numbers, which they acquire via the socket API. As we discussed earlier in Chapter 3, process-to-process addressing requires makes use of network sockets. A socket is identified by a combination of IP number(s) and port number(s).

A port number is a 16-bit integer that takes a value between 0 and 65535. Internet Assigned Numbers Authority (IANA), which is a sub-department of Internet Corporation for Assigned Names and Numbers (ICANN), is the official authority that 'reserves' port numbers to various applications. The port numbers ranging from 0 to 1023 (also known as *well-known* port numbers) are reserved for applications such as FTP (file transfer), HTTP (world-wide web) and SMTP (e-mail).

Q&A on process addressing and the use of sockets

You may read in different places different usages of port numbers and sockets. The following gives a list of questions and 'standard' answers regarding

port numbers and network sockets. Note that even though this may not be 'good practice', it is not strictly forbidden to do it differently and you may find operating systems that allow alternative ways.

- A destination port number identifies a process on a destination host. Is it true always that each process has one unique port number?
  - Two processes may listen on the same port number. For example, a web server can run multiple processes listening on the port number 80. When a transport layer segment with destination port number 80 arrives to this server, the extracted data is sent to one of these web server processes by the transport layer.
- Is there a single network socket per process or can a process use more than one socket?
  - There can be more than one socket per process, each with a unique identifier. The exact format of the identifier depends on whether the socket is a UDP or TCP socket.
- Can a destination process receive packets from multiple source hosts on a single socket?
  - Yes, this is possible with UDP. TCP does not allow this as it is based on a 'connection' with only one sender and one receiver (see Section 4.3).
- How does IANA enforce the use of well-known port numbers explicitly by certain application types?
  - Well-known port numbers being 'reserved' does not mean that it is impossible to use them in other applications. That would be against the layering philosophy. The reservation helps to standardize servers. The developers of other applications can consciously stay away from well-known port numbers in order to avoid confusion.

4.3 Connection and Connection-less Service

The transport layer can provide a connection-oriented service or a connection-less service.

In connection-oriented service data transfer is necessarily between processes that have first established a connection. A connection is similar to a pipe whose only entry points are at the two ends (i.e. where the network sockets are). The 'connection' allows the communicating processes to use the notion of a packet stream through a virtual pipe. Other processes outside of the connection cannot send packets to the sockets of a connection. We say that a connection-oriented service is stateful. Remembering the state is, for example, through remembering the connection status, the order in which packets are sent and the list of packets that are already received. TCP is an example of a connection-oriented transport layer protocol. For example, a web browser makes a TCP connection to a web server process before sending it HTTP requests. Once the socket is created, it

is a one-to-one communication and another web browser may not make HTTP requests to the same socket (but it may make HTTP requests to the same process through another TCP socket).

A connection between the processes is established using a message exchange procedure called a 'three-way handshake' as shown in Figure 4.4.[2] A connection is terminated as shown in Figure 4.5.

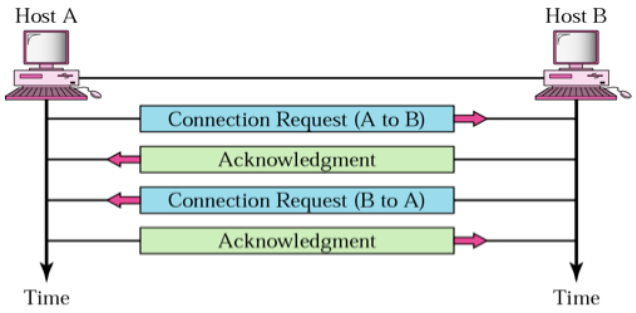


Figure 4.4: Three-way handshake for connection establishment in connection-oriented service. (Figure by Forouzan)

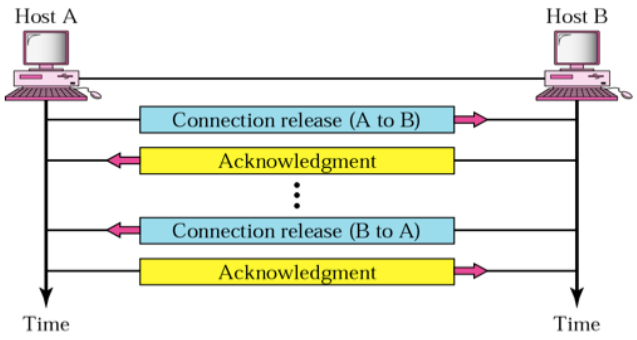


Figure 4.5: Connection termination in connection-oriented service. (Figure by Forouzan)

A connection-less service on the other hand does not require the establishment of a connection prior to data transfer and it is stateless. Processes are free to send data packets to any connection-less socket and all will be delivered to the receiving process through the socket. UDP is an example of a connection-less transport layer protocol.

[2]You may ask yourself why three-way handshake when there are a total of 4 messages. It is because the second and the third messages are typically merged into one message. In full-duplex communication where data packets flow in both directions, sending acknowledgement data together with application data is called *piggybacking*.

4.4 Internet's Layer 4 Protocols: UDP and TCP

The two transport layer protocols that are used in the Internet are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Application developers have to make a choice between these two protocols and they do it based on the specific requirements of the application being built. We already discussed (Figure 4.1) some application requirements. Figure 4.6 shows some example Internet applications, their respective application layer protocols and the transport layer protocol chosen to fulfill the application requirements.

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g. Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Figure 4.6: Internet applications, application layer protocols and underlying transport layer protocols. (Figure by Kurose and Ross)

4.4.1 UDP

UDP is a connection-less transport layer protocol that provides multiplexing and demultiplexing as the only services. Although UDP lacks most of the services that TCP provides, UDP applications enjoy advantages of its simplicity:

- UDP packet header size is only 8 bytes and that means very small communication overhead in comparison to 20+ bytes long TCP headers. This is especially important if the application needs to communicate very little data (a few bytes) very often.
- UDP does not perform congestion control, which means that UDP applications can utilize the full speed of the communication channel.

A UDP socket is identified by a 2-tuple: the destination IP address and the destination port number. Since there is no notion of connection in UDP, all segments that have the same destination IP and port number will be directed to the same UDP socket (hence the same process). This is shown in Figure 4.7.

Definition 4.4.1.1 *Word*

The term 'word' is used to refer to a fixed-length sequence of bits. The length of a word is naturally chosen according to the unit of computation in a certain computer architecture (e.g. 32-bit words for 32-bit computers).

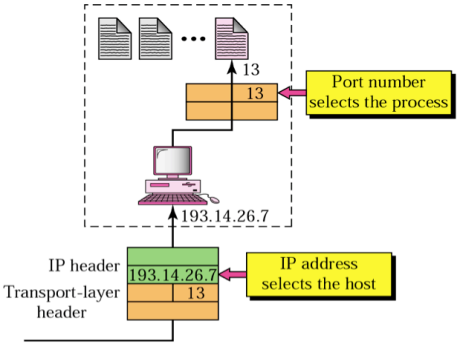


Figure 4.7: The destination process is addressed using the destination IP address and the destination port number. (Figure by Forouzan)

UDP packet format

In this course, we will illustrate packet formats in lines that divide the packet into 32-bit words. The UDP segment format is given in Figure 4.8. The header consists of four header fields. The first word contains source and destination port numbers (16 bits each, integers ranging between 0 and 65535). The second word contains the total packet length including header (in number of bytes) and the checksum. The checksum is used for a simple check of integrity, i.e. whether there are bit errors (changed bits) in a received segment or not.

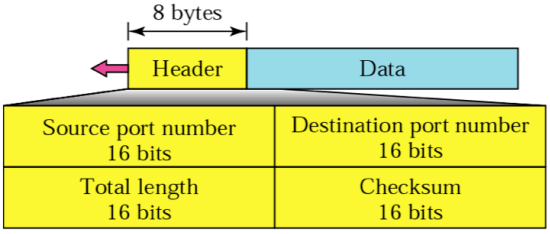


Figure 4.8: UDP packet format. (Figure by Forouzan)

The UDP sender computes the checksum over certain parts of the IP header (called the *pseudo IP header*), the UDP header and the payload. The computation considers all this as a sequence of 16-bit integers and adds them up (1's complement sum). The checksum is the 1's complement of the result. In order to check for errors, the UDP receiver simply has to compute the checksum of a received packet again and compare it with the checksum header field value. If these two values are not equal, the receiver concludes that the segment is corrupt and discards the segment. Note that this method may still miss some errors, i.e. when two or more bits are flipped.

4.4.2 TCP

TCP is a connection-oriented transport protocol that provides extra services on top of multiplexing and demultiplexing such as

- connection setup, maintenance and termination
- reliable data transfer
- flow control
- congestion control

TCP by itself does not provide timing, throughput or security guarantees. Security can be implemented on top of TCP in a layer 4.5 protocol such as Transport Layer Security (TLS).

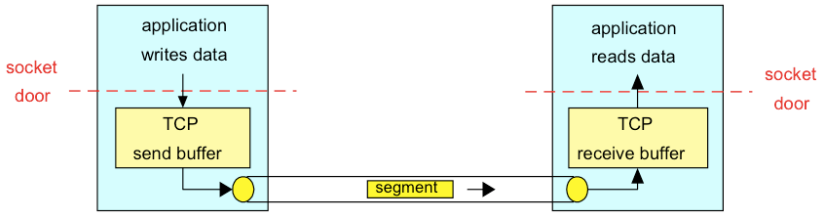


Figure 4.9: TCP sockets, buffers and the (byte stream) connection between two processes. (Figure by Kurose and Ross)

Making a TCP connection between two processes requires a connection establishment procedure. A TCP socket is identified by the following 4-tuple and **defines a connection**:

- source port number
- destination port number
- source IP address
- destination IP address

In demultiplexing, the receiver uses this full identifier consisting of four parts to choose the correct delivery socket (hence the correct process). Because of this, for example a web server is able to deal with many TCP connections from different users at port number 80. Since the source IP addresses and/or source port numbers of the web browsers of individual users are different, their TCP segments are delivered to different TCP sockets with identical destination IP and destination port number fields. This is illustrated in Figure 4.10 using the web server example.

Even for the same user, (non-persistent) parallel HTTP connections will result in a different socket (identifier) for each request since parallel TCP connections will have different source port numbers. This is illustrated in Figure 4.11.

TCP packet format

The TCP packet format is shown in Figure 4.12.

- The first word contains source and destination port numbers.

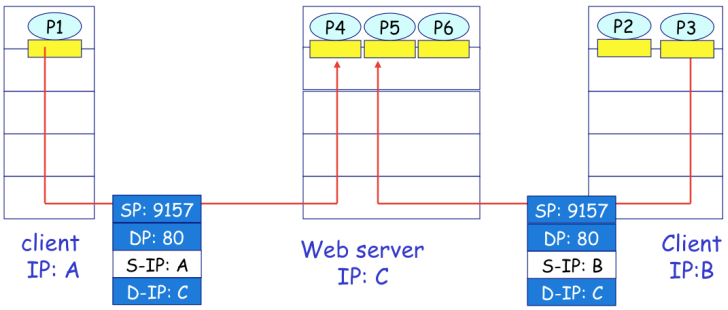


Figure 4.10: TCP demultiplexing for segments with identical header fields except for sender's IP address. (Figure by Kurose and Ross)

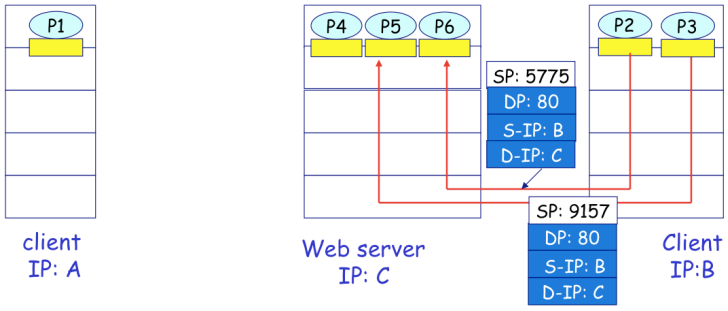


Figure 4.11: TCP demultiplexing for segments with identical header fields except for sender's port number. P2 and P3 together could have been a single process but that would not change anything. (Figure by Kurose and Ross)

- The second word contains the sequence number, which is in fact the index of the first byte of the segment within the whole byte stream. Note that the numbering is on the byte count and not the segment count. Therefore, consecutive segments do not necessarily have sequential numbers. Furthermore, the sequence numbers in each direction are independent and both start from a random value. Numbering is used for error control and flow control.

- The third word contains the acknowledgement number, which confirms the reception of bytes to the sender. This field indicates the index of the next byte in the byte stream that is expected by the receiver. By filling this field with the number N, the receiver says to the sender that it is now expecting byte number N of the byte stream and it has received **all** bytes prior to this (i.e. **acknowledgements are cumulative**).

- The fourth word starts with the HLEN field that indicates the total length of the TCP header in number of words. This is followed by 6 bits that are reserved[3] for future use. The window size field indicates the number

[3]Three of the six are already standardized in the meantime. Interested students can check RFC 3540 and RFC 3168 for more details.

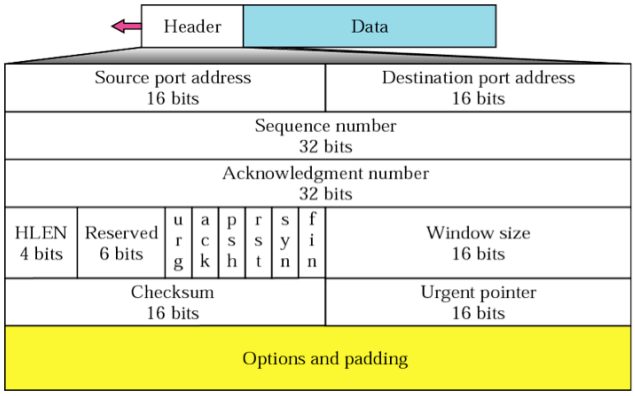


Figure 4.12: TCP packet format. (Figure by Forouzan)

of bytes that the receiver is willing to receive and is used for flow control. The six 1-bit flags are:

  - URG: 1 if the urgent pointer field is to be used.
  - ACK: 1 if acknowledgment field is used.
  - PSH: 1 if the sender wants the receiver to push the already buffered data portion to the receiving process.
  - RST: 1 to reset the TCP connection.
  - SYN: Used while establishing a connection (1 for the first packet of the byte stream).
  - FIN: Used for terminating an active connection.

- The fifth word contains checksum and urgent pointer fields. The checksum use is the same as in UDP. The urgent pointer field is used if the URG flag is set and it is an offset from the sequence number indicating the last urgent data byte.

- The first five words of the header are followed by the 'Options and padding' fields, which can be up to 10 words long (HLEN determines the total length). We will not discuss these. Padding refers to filling with zeros until the total header length is an integer multiple of 32 bits.

Reliable Data Transfer and TCP

TCP provides reliable transfer of data between two processes. Note that the reliability of data transfer *i*) between end hosts and *ii*) between neighboring nodes may also be services provided by the network and the link layers below, respectively. At this point in our discussion we will assume that the underlying network layer and the link layer are unreliable.

TCP provides reliability using source-to-destination retransmissions that rely on the following mechanisms:

The receiver side uses buffering, sequence numbers, TCP checksum and cumulative acknowledgements for contributing to reliable data delivery.

- The TCP receiver detects and discards segments that are corrupted (using checksum) or that are duplicates (using sequence numbers).
- The receiver confirms the reception of bytes to the sender side by sending cumulative acknowledgements. Precisely, the receiver acknowledges the sequence number of the next byte that is expected. The events leading to acknowledgement generation by the receiver are given in Figure 4.13.

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

Figure 4.13: TCP events leading to acknowledgements. (Figure by Kurose and Ross)

- The receiver buffers the segments that are received out-of-order and delivers data to the destination process only after filling in the gaps (i.e. data arrives at the process in order). When segments arrive at the receiver out-of-order, the receiver acknowledges the first byte of the gap (at the lower end). Therefore, repetitively receiving out-of-order packets that do not contain the 'next expected byte' will trigger duplicate acknowledgements.

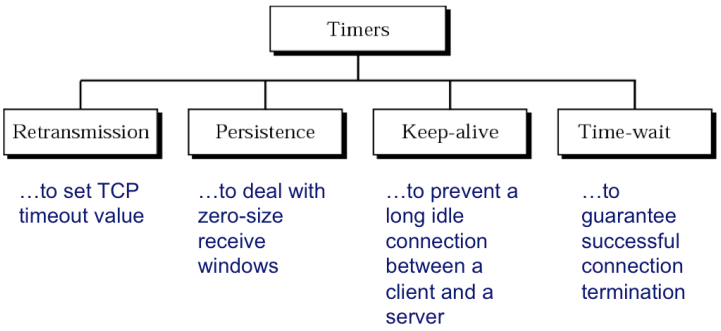


Figure 4.14: Four types of TCP timers. (Figure by Forouzan)

The sender side relies on TCP retransmission timer and counting duplicate acknowledgements for making a decision to retransmit segments or not.

- The TCP sender maintains four timers (Figure 4.14). One of these timers is the **retransmission timer**. If an acknowledgement confirms the reception of previously unacknowledged packets, this timer is either stopped or reset (see the exact operation of the TCP sender with retransmission timer and acknowledgements in Figure 4.15). TCP interprets expiration of this timer as an indication of congestion. When the timer expires, the TCP sender retransmits the segment and resets the timer. Figure 4.16 gives examples of TCP retransmission scenarios.

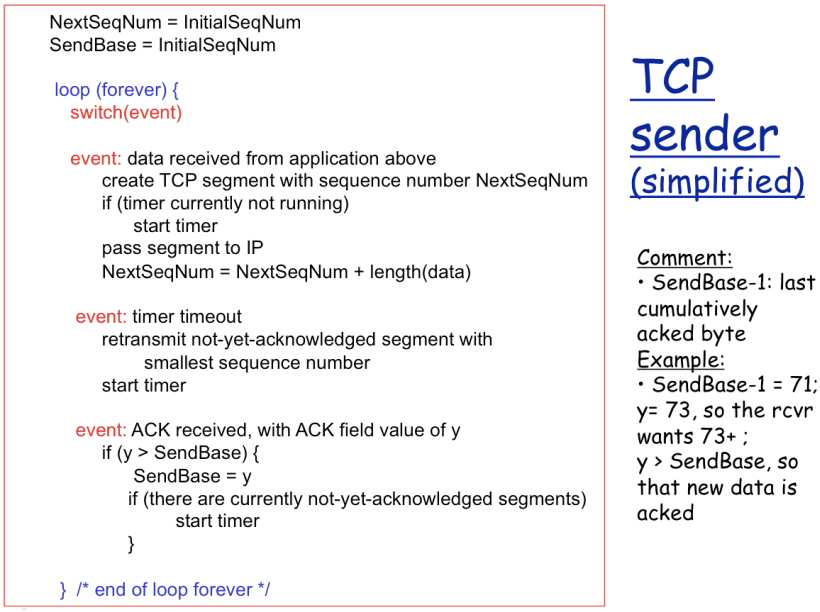


Figure 4.15: Simplified TCP sender operation (ignoring duplicate acknowledgements, without congestion and flow control). (Figure by Kurose and Ross)

- **Fast retransmit:** The retransmission timer may take a long time to expire, resulting in late retransmissions of lost packets. The TCP sender continuously checks the acknowledgement numbers filled in by the receiver and tries to identify packet losses by looking at duplicate acknowledgements. The logic of 'fast retransmit' is simple: The sender typically pipelines its segments and sends many segments in a short time. If there is congestion, it is likely that a lot of these segments will be lost and the sender will not get many (duplicate) acknowledgements. However, if it is just a single segment (or a few) that is lost, there will be many duplicate acknowledgements (asking for the first byte of the gap). When the TCP sender receives 3 acknowledgements pointing to the first byte of the lost segment, it will go into the fast retransmit mode and retransmit this segment. The necessary modification to the TCP sender to include fast retransmit is shown in Figure 4.17, together with an example.

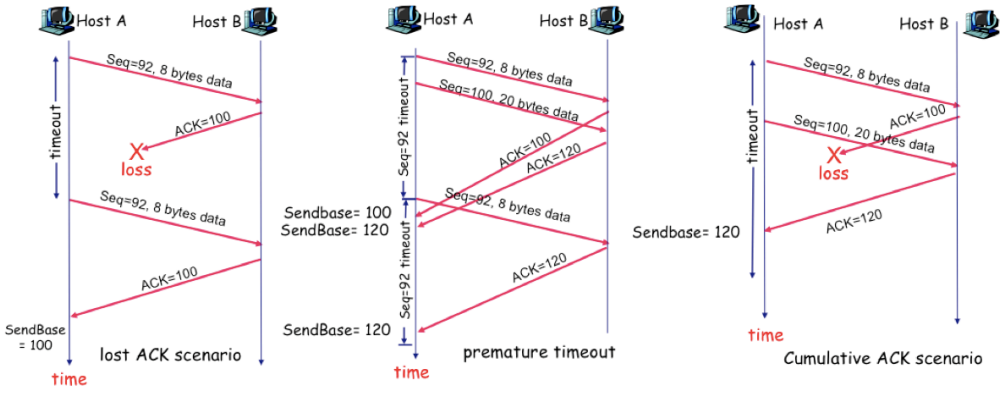


Figure 4.16: TCP retransmission examples. (Figure by Kurose and Ross)

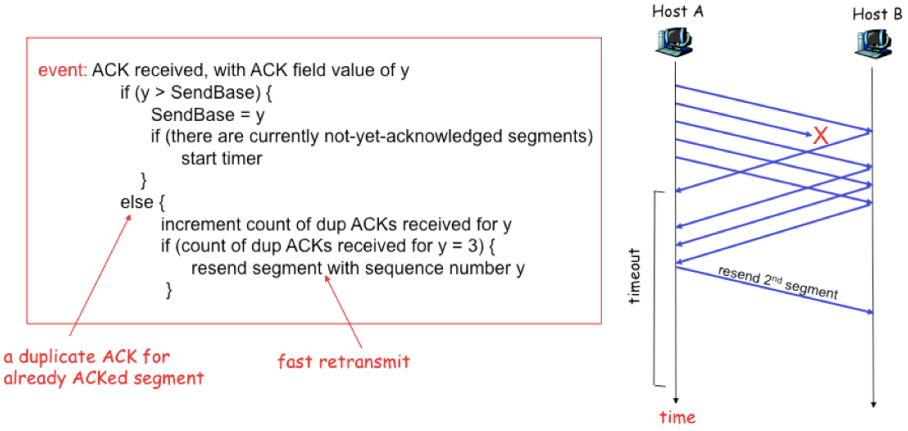


Figure 4.17: Simplified TCP sender operation with fast retransmit (without congestion control and flow control). (Figure by Kurose and Ross)

TCP Flow Control

The Internet is an global heterogeneous network with hosts and access links working at different speeds. Imagine that an application on your smart watch is downloading a large file (e.g. an operating system update). The download throughput that is available can be much more than that can be handled by the smart watch, for example, if writing to the secondary storage is slow on the smart watch. What is needed for correct operation is a bitrate (bits per second) matching service between the sender process and the receiver process. This is more or less what the flow control of TCP does.

The receiver fills in the **window size** field of each TCP segment (i.e. in pure acknowledgements or in data segments in two-way communication) the number of bytes that it is still willing to receive it its buffer (see Figure 4.18). The sending process uses this information to adjust its speed. A window size of zero causes the sender to stop sending data and start the **persistence timer**. When this timer expires, the sender sends one (small) packet to probe the status of

the receiver window size and avoid deadlock.

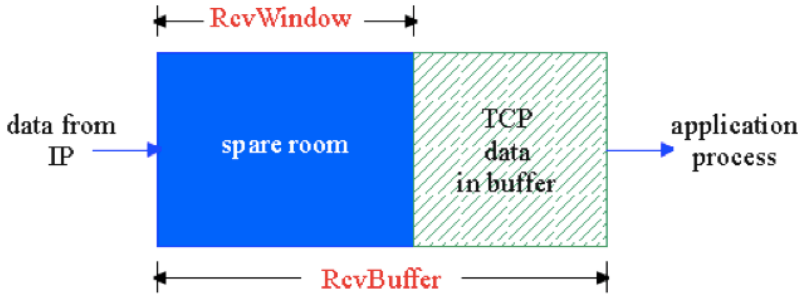


Figure 4.18: The receiving process may be slow at reading from this buffer. (Figure by Kurose and Ross)

TCP Congestion Control

Congestion is a result of increasing network traffic, i.e. the number of packets pumped into the network getting close to the overall network capacity or, even worse, exceeding it. We discussed in Chapter 1 that the router queues grow in this case and a large fraction of packets either experience very long queuing delays and or have to be dropped. Congestion control is an effort to avoid this.

The TCP sender assumes that the cause of a retransmission timeout is congestion and consequently slows down. One sender throttling down its transmission rate will not make a huge effect, but the collective effort of many senders can make a difference. After slowing down, the sender again gradually increases its transmission rate until the next retransmission timeout.

Slow Start and Additive Increase: The TCP sender starts with a **congestion window (cwnd)** that is equal to 1 segment (with size **maximum segment size (MSS)**[4]). At this **slow start** phase, starting from a cnwd of 1 the sender is trying to go up to a large congestion window very quickly. To do this, the sender increases its congestion window size by 1 segment until cwnd reaches a threshold value (effectively an **exponential increase** of speed until the threshold). At this point the sender goes into a **congestion avoidance** phase where cwnd is increased by 1/cwnd segments for each successfully received acknowledgement until cwnd is equal to the receiver window size.

Multiplicative Decrease: Upon a retransmission timeout the cwnd size is reset to 1 segment and the threshold is set to cwnd/2. In **TCP Tahoe**, when three duplicate ACKs are received, the sender does a fast retransmission and goes back into the slow start phase with the threshold set to cwnd/2. In **TCP Reno**, when three duplicate acknowledgements are received, the sender does a fast retransmission and goes into the **fast recovery** phase with the threshold set to cwnd/2, skipping slow start. In fast recovery, cwnd is increased by 1/cwnd

[4]Depends on size of the largest IP datagram that the host can handle.

segments for each successfully received acknowledgement until cwnd is equal to the receiver window size.

This algorithm applied by the TCP congestion control is called **Additive Increase, Multiplicative Decrease (AIMD)** and is shown in Figure 4.19.

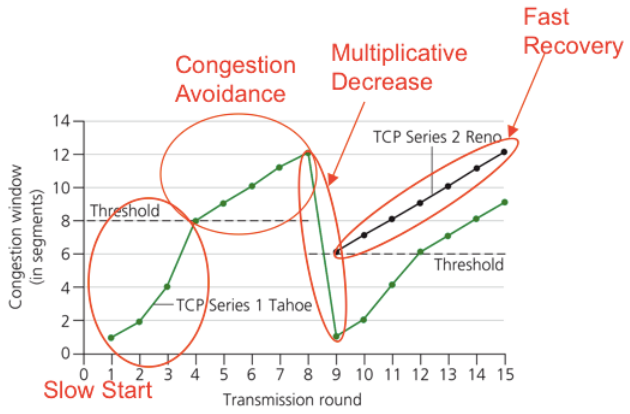


Figure 4.19: TCP applying AIMD. (Figure by Kurose and Ross)

At any given time during transmission the send window of the sender (i.e. maximum number of in-flight segments) is equal to the minimum of the receiver's advertised window size and the congestion window size cwnd.

TCP Vizualizer

Visit `http://www.win.tue.nl/~mholende/tcpviz/` in order to download **TCP vizualizer** software tool that is developed and maintained by Dr. Mike Holenderski.

TCP Visualizer aims at showing the differences between three flavours of the TCP protocol: TCP *Tahoe*, TCP *Reno* and TCP *New Reno*. It shows different views on the TCP protocol from the perspective of the sender:

- the outgoing buffer together with the sliding congestion window (CWND)
- a plot of the CWND size and threshold (SSTHRES) against time and against round-trip time (RTT)
- a plot of sent DATA segments and received acknowledgement (ACK) segments against time
- a sequence diagram describing occurring events (e.g. arrival of an ACK segment or a timeout) and the corresponding actions

4.5 Summary

Transport layer provides services on top of the network layer. The most basic transport layer service is process addressing and process-to-process data delivery. UDP provides only these basic services while TCP provides additional services,

namely, connection setup, maintenance and termination, reliable data transfer, flow control and congestion control. TCP performance in terms of throughput and delay can be hampered by congestion control while UDP fires as fast as it can. In practice, UDP-based applications should be 'TCP friendly'.

4.5.1 Literature

Suggested reading

- Chapter 3 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 3].

4.6 Homework Exercises

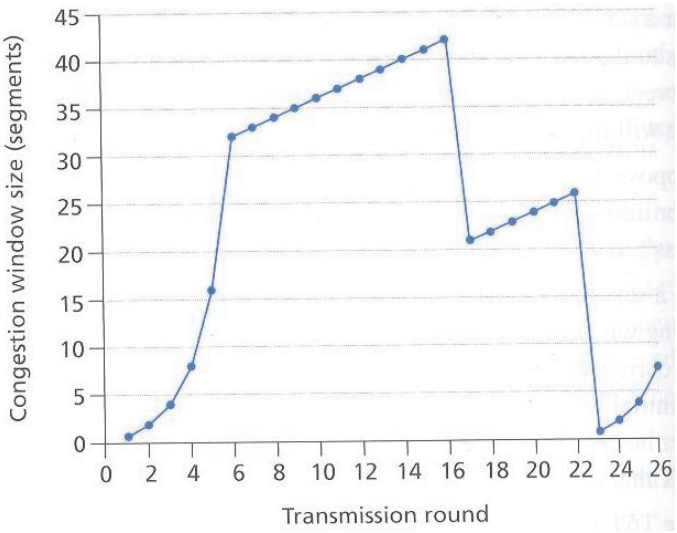
1. What is the need for UDP? Would it not have been enough to just let user processes send raw IP packets? Explain in a few sentences.

2. Consider sending a large file from one host to another over a TCP connection that has no loss.

   (a) Suppose TCP uses AIMD for its congestion control without slow start. Assuming CongWin increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times, how long does it take for CongWin to increase from 1 MSS to 8 MSS (assuming no loss events)?

   (b) What is the average throughput (in terms of MSS and RTT) for this connection up to time = 7 RTT?

3. **Transmission Control Protocol (TCP)-** Consider the following plot of TCP window size as a function of time.



Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.

(a) Identify the intervals of time when TCP slow start is operating.

(b) Identify the intervals of time when TCP congestion avoidance is operating.

(c) After the 16^{th} transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

(d) After the 22^{nd} transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

(e) What is the initial value of `Threshold` at the first transmission round?

(f) What is the value of `Threshold` at the 18^{th} transmission round?

(g) During what transmission round is the 70^{th} segment sent?

(h) Assuming a packet loss is detected after the 26^{th} round by the receipt of a triple duplicated ACK, what will be the values of the congestion control window size and of `Threshold`?

4. Suppose that a 1Mbps connection is shared by 5 well behaving TCP and 5 UDP sources. The maximum window size of the TCP sources is 50 KB. UDP sources send at a speed of 100 Mbps. What is the maximum transmission speed of UDP and TCP sources? Justify your answer.

5. Suppose Host A sends two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 90; the second has sequence number 110.

   (a) How much data is in the first segment?

   (b) Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgment number?

6. Reliable data transfer:

   (a) Consider a reliable data transfer protocol that uses only negative acknowledgments (NAKs). Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses acknowledgments (ACKs) only? Why?

   (b) Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

Chapter 5

Network Layer

In this chapter, we try to answer the following questions:

- What are the principles of network layer services and protocols?
- What are the details of IP (Internet Protocol)?
- How are addressing and routing are done in a network, especially in a large network like the Internet?

The main responsibility of a network layer protocol is host-to-host data delivery. In doing this, the network layer has to deal with network layer addressing, packetization and fragmentation, routing (and forwarding), as well as the concept of internetworking (connecting links of various types into a united internetwork view). There are two logical types of packet-switched networks: *datagram networks* and *virtual circuit networks*.

The **Internet Protocol (IP)** governs the network layer of the Internet. The 'IP layer' is between the transport layer and the link layer. It provides services to the transport layer and it receives the services of the link layer (e.g. node-to-node delivery) as shown in Figure 5.1.

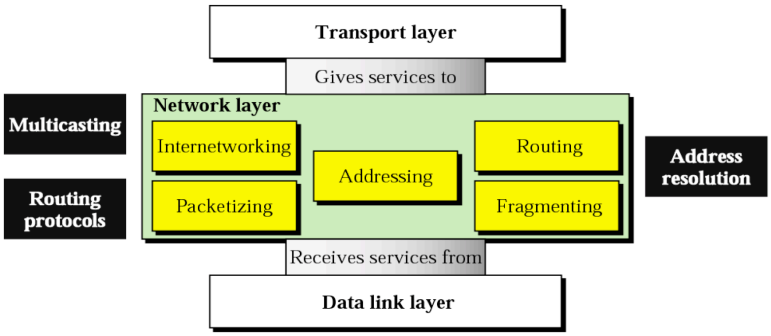


Figure 5.1: Issues addressed by network layer protocols. (Figure by Forouzan)

5.1 Internetworking

The host-to-host path in a computer network may pass through many types of (wired and wireless) physical network links, and the **internetworking** service provided by the network layer is an abstraction from this diversity. Furthermore, despite the variety of link technologies and the link-specific addresses and protocols, the transport layer enjoys a uniform numeric **addressing** scheme provided by the network layer.

The network layer of the Internet is implemented in both the network edge (hosts: clients, servers, peers) and the network core (routers) as shown in Figure 5.2.

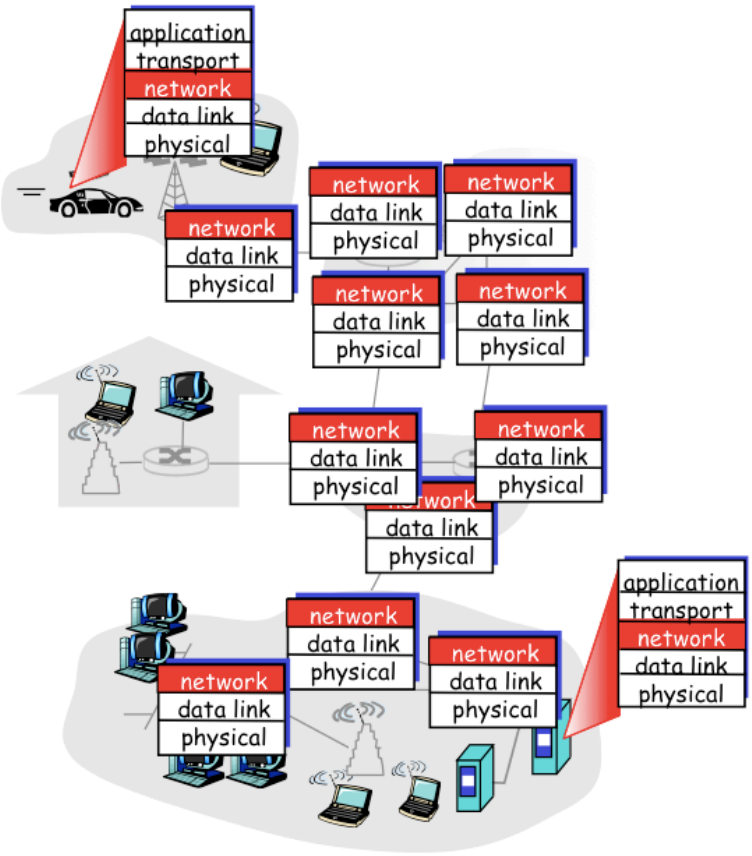


Figure 5.2: Network layer protocol is implemented in every router and host. (Figure by Kurose and Ross)

> *"Internetworking is the practice of connecting a computer network with other networks through the use of gateways that provide a common method of routing information packets between the networks."*
>
> Wikipedia

In the Internet setting, the routers in the network core interconnect **subnetworks** (also known as subnets) and they correspond to the gateways mentioned in this definition.

5.2 Datagram Networks

This course focuses on computer networks that are **packet-switched** where data are communicated in bit chunks that are called **packets**. Packet-switched networks that provide a connection-less unreliable service (e.g. the Internet) are called **datagram** networks.

In datagram networks, there is no concept of a session state in the network core[1]. Routers treat each datagram independently and their per-datagram behavior is statistical (destination network address is determining in choosing the outgoing interface). As a result, different datagrams traveling from sender to receiver in a datagram network can take different paths as shown in Figure 5.3.

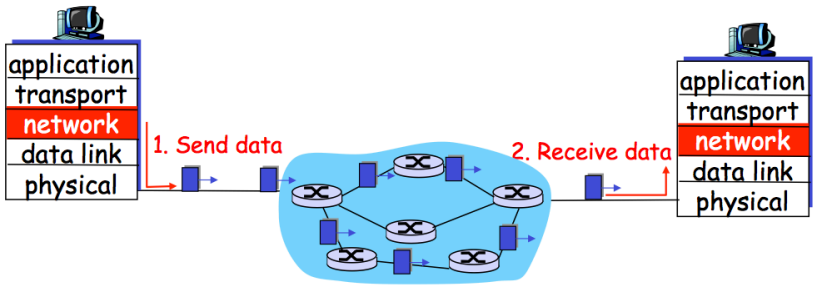


Figure 5.3: Different datagrams traveling from sender to receiver in a datagram network can take different paths. (Figure by Kurose and Ross)

5.2.1 Packetization

Network layer packetization refers to the encapsulation of transport layer segments into network layer datagrams by the sending host and the delivery of these segments to the transport layer by the receiving host.

IP version 4 (IPv4)

The packet format of IPv4 datagrams is shown in Figure 5.4.

VER: IP version (v4).

HLEN: IP header length in number of 32-bit words.

DS: Differentiated services code indicating the type of data in the IP datagram.

Total length: Total length of the datagram in number of bytes.

[1]Note that sessions may still exist at higher layers, but the network core does not give support for that.

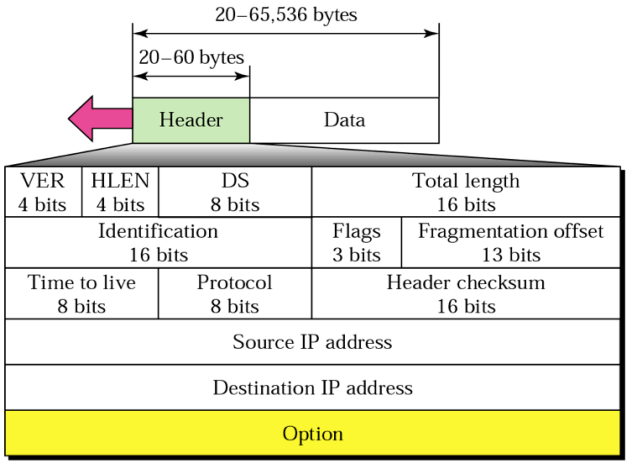


Figure 5.4: IPv4 datagram format. (Figure by Forouzan)

Fragmentation fields:

- **Identification:** Identifier for the fragments of a single datagram.
- **Flags:** Fragmentation controls and bit flag for indicating the last fragment of a datagram.
- **Fragmentation offset:** The fragment offset (multiples of 8 bytes starting from zero).

Time To Live (TTL): The number of hops (links) that the datagram is still allowed to take (value decremented at each router).

Protocol: Target layer 4 protocol for the payload of the datagram.

Header checksum: 16-bit checksum of the IP header as shown in Figure 5.5. The payload is not included in the checksum calculation.

Source and destination IP addresses: IPv4 addresses are 32-bit network layer identifiers of link interfaces of Internet hosts and routers. IPv4 addresses use the dotted decimal notation as shown in Figure 5.6.

Definition 5.2.1.1 *Link Interface*

A link interface of a network node is its physical connection the corresponding physical link.

Since routers have multiple link interfaces they have multiple IP addresses, one for each interface. A significant portion of modern hosts also have multiple link interfaces (e.g. one Ethernet, one WiFi) and multiple IP addresses. Using 32 bits we can individually address 2^{32} (roughly 4 billion) IPv4 node interfaces, which is insufficient in our day.

Option: This part contains extra information fields that are mostly skipped. We will not study those in this course.

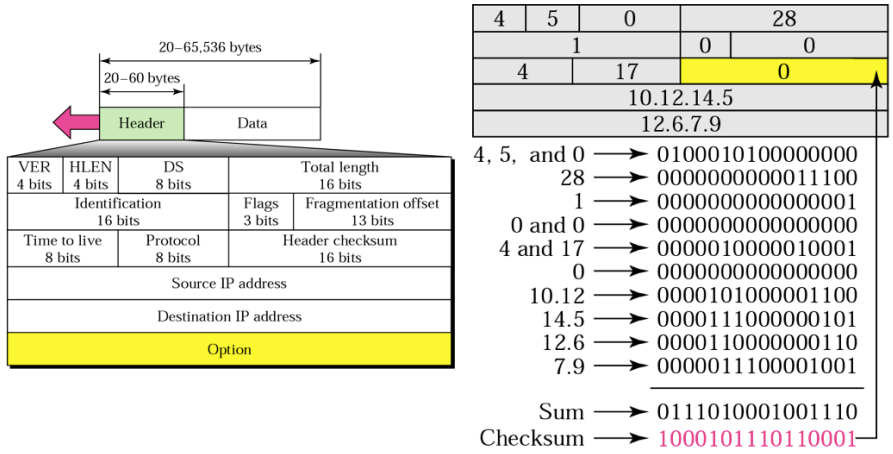


Figure 5.5: IPv4 checksum calculation. (Figure by Forouzan)

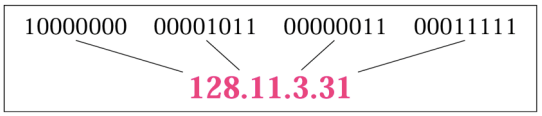


Figure 5.6: IPv4 dotted decimal notation. (Figure by Forouzan)

IPv4 Fragmentation

The maximum size of the payload that can be encapsulated within a data link layer frame is dependent on the specific link type. This size is referred to as the Maximum Transmission Unit (MTU). For example, Ethernet has an MTU of 1500 bytes while WiFi (wireless IEEE 802.11 protocol) has an MTU of 2400 bytes. Upon transmitting a datagram on the outgoing link, if the size of the IP datagram is larger than the link MTU, the transmitter (i.e. a source host or a router on the path) needs to divide the datagram into pieces that can fit in a frame. This is called **fragmentation**. After fragmentation, individual fragments travel to the destination host independently from each other and they are reassembled into the full IP datagram only at the last destination host.

The second word (second line) of the IPv4 header contains fields that are dedicated to this purpose as shown in Figure 5.4.

IP version 6 (IPv6)

IPv6 was developed as a result of the 32-bit IPv4 address space being completely exhausted. IPv6 addresses consist of 128-bits, which means that there are 2^{128} IPv6 addresses. Just to put this amount in perspective, this is (way) more than enough to give an IPv6 address to every piece of sand in the entire world.

The packet format of IPv6 is simpler than the IPv4, which speeds up processing at routers. The packet format is shown in Figure 5.9. The 'option' fields have been removed from the basic header but this type of information (indicated by the 'Next header' field) can now be part of the payload. In IPv6, routers

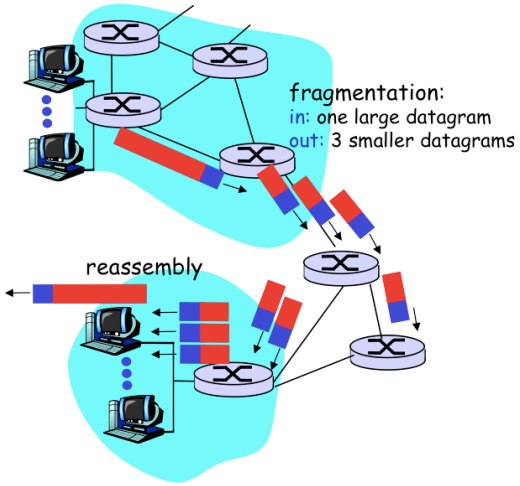


Figure 5.7: IPv4 datagram that is fragmented when moving towards link with smaller MTU: Different fragments may take different paths and they are reassembled in the destination host. (Figure by Kurose and Ross)

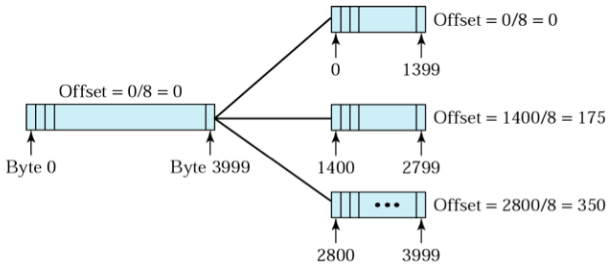


Figure 5.8: IPv4 fragmentation. (Figure by Forouzan)

do not deal with fragmentation. Sending hosts are required first to decide on a datagram size using MTU discovery along the host-to-host path.

One of the goals of the IPv6 original design was to add security features to IP. The **Internet Protocol Security (IPsec)** is a set of protocols for authentication and encryption of IP packets. It was originally designed for IPv6.

VER: IP version (v6).

PRI: Priority field used to classify packets.

Flow label: Labels packets belonging to the same flow with the same number. Routers can use this information so that the packets take the same path and maintain the same order.

Payload length: The payload length in number of bytes.

Next header: Specifies the transport layer protocol or the type of payload extension headers that follow.

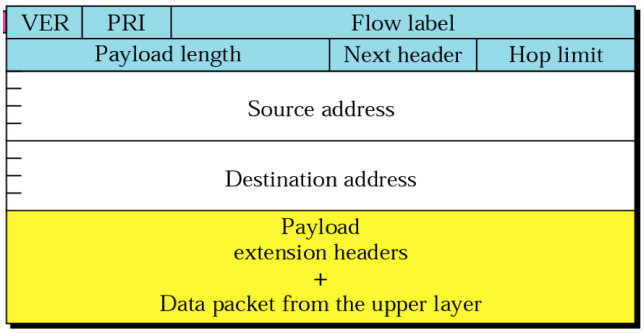


Figure 5.9: IPv6 datagram format. (Figure by Forouzan)

Hop limit: The same as TTL of IPv4.

Source and destination IP addresses: An IPv6 address is represented using hexadecimal colon notation (each set of 4 hexadecimal digits correspond to 16 bits) as shown in Figure 5.10.

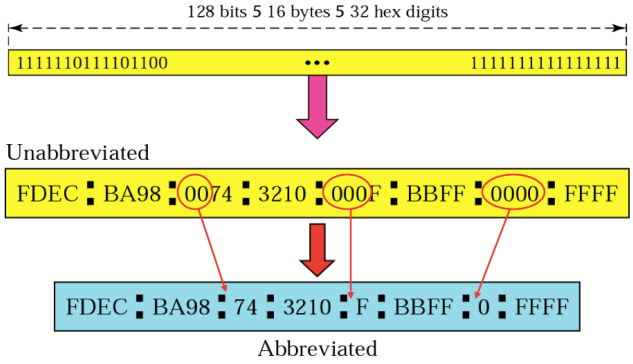


Figure 5.10: IPv6 address representation. (Figure by Forouzan)

5.3 Layer 3 Addressing and Subnetting

A subnetwork or subnet in an internetwork is a set of network interfaces that can physically reach each other over a single link (no routers in between). In order to identify subnets in a network, a simple recipe is to detach each router interface resulting in islands of isolated networks, each being a subnet. An example is shown in Figure 5.11.

In the first years of the Internet, the network layer divided the total address space (consisting of 2^{32} addresses) into classes as shown in Figure 5.12, where the first few bits indicate the class of the IPv4 address. Furthermore, each IPv4 address consisted of network identifier and host identifier fields as shown in Figure 5.13.

Classful addressing has efficiency issues. For example, assume that we want to assign an address space to an organization (e.g. a university) with 2000 hosts.

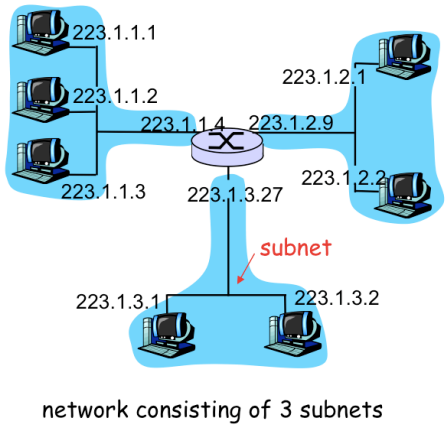


Figure 5.11: A network with three subnets. (Figure by Kurose and Ross)

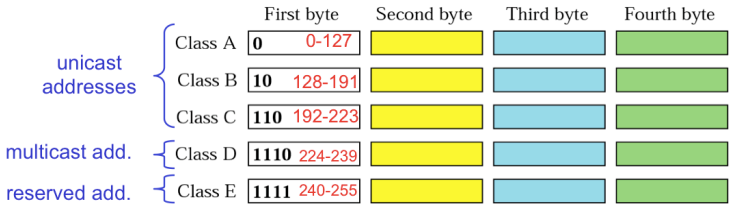


Figure 5.12: Classful IPv4 addressing. (Figure by Forouzan)

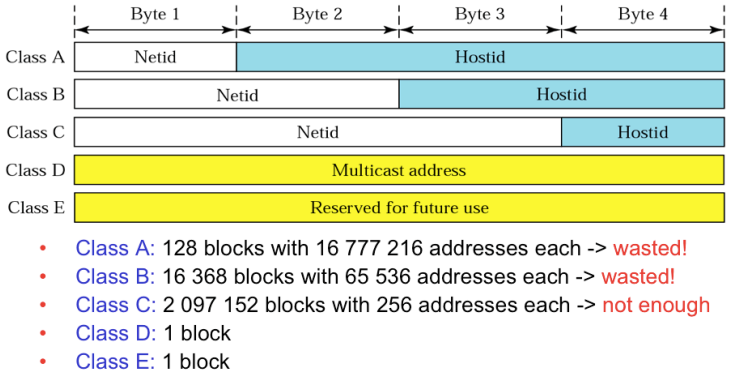


Figure 5.13: Classful IPv4 addressing: Netid and Hostid parts of different classes. The example shows the efficiency of assigning different class address spaces to an organization with 2000 hosts. (Figure by Forouzan)

Assigning class B would result in a waste of more than 63K IPv4 addresses and we can make class B assignments to at most 16368 such organizations. On the other hand, assigning class C is not possible since class C gives only 256 distinct addresses in each block. A workaround for this problem would be to assign multiple class C blocks to the organization.

Due to inefficiencies of classful addressing, classless addressing and Classless Inter-Domain Routing (CIDR) were introduced. In classless addressing an IP address consists of a subnet part (higher order bits) and a host part (lower order bits). The subnet parts of all IP addresses in a given subnet are identical, i.e. all these IP addresses start with the same **subnet prefix**. We illustrate the notations used for subnet prefix and subnet mask in Figure 5.14 (IPv6 is similar).

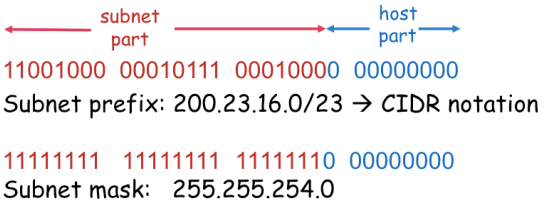


Figure 5.14: CIDR notation: Subnet part and host part of an IPv4 address. (Figure by Kurose and Ross)

Subnetting maps an address to the host's location in a network topology. That is, hosts inside a subnet do not own their IP addresses by default. Newly joining hosts can 'lease' their IP addresses from a server for a certain time duration using the Dynamic Host Configuration Protocol (DHCP) as shown in Figure 5.15.

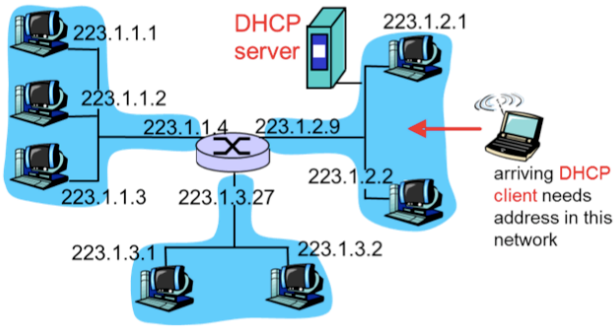


Figure 5.15: DHCP: New host needs IP in the top right subnet. (Figure by Kurose and Ross)

Since used IP addresses eventually return to the pool of available IP addresses[2], DHCP allows reuse of IP addresses in a subnet. Upon receiving the IP address, the DHCP server also provides additional information about the subnet mask, the default gateway, the local DNS server address etc.

In DHCP, a newly joining host acquires an IP address as follows:

1. The new host broadcasts a "DHCP discover" message.
2. The DHCP server responds with a "DHCP offer" message.

[2]The maximum pool size is limited by the ISP's allocated block of IP addresses for this subnet.

3. The host requests an IP address by sending a “DHCP request” message.

4. The DHCP server assigns and sends the IP address in a “DHCP ack” message.

5.4 Network Layer Routing

In its host-to-host path a network layer packet goes through multiple routers and subnets. A **routing protocol** determines the packet routing behaviors of individual routers in a network. The routing protocol implementation at routers builds router **forwarding tables**, i.e. a mapping from destination host addresses to the outgoing link interfaces of the router. While forwarding concerns moving packets from an incoming link of the router input to the correct outgoing link, routing concerns determining the entire route taken by packets from source to destination as shown in Figure 5.16.

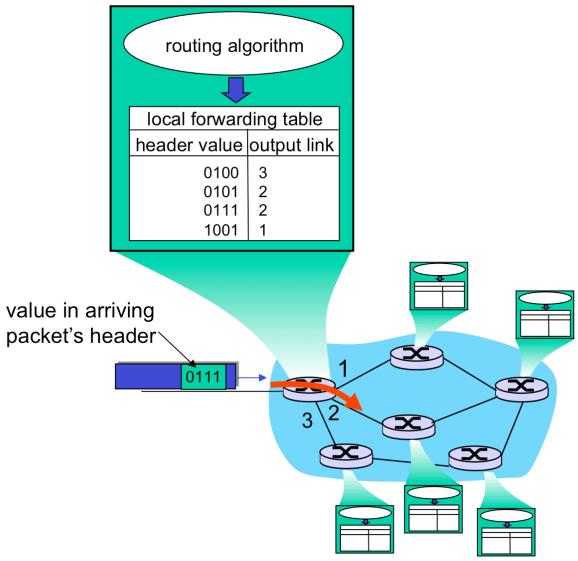


Figure 5.16: The routing algorithm determines the forwarding tables of the routers. (Figure by Kurose and Ross)

Figure 5.17 shows how a forwarding table of a router could look in principle, with one entry per destination IP. However, such a forwarding table is impossible to maintain and use for a simple reason. Even with IPv4 addresses consisting of 32 bits there would need to be around 4 billion entries in such a forwarding table and every forwarding operation would require a look-up among these entries. Furthermore, the amount of information exchange of the routing protocol for maintaining forwarding tables of routers would cause a constant state of congestion, rendering the Internet unusable for any other data communication.

Internet addressing is hierarchical. Hosts lease IP addresses from their subnets. The network administrator of the subnet can acquire a block of IP addresses for the subnet from an Internet Service Provider (ISP). The ISP itself

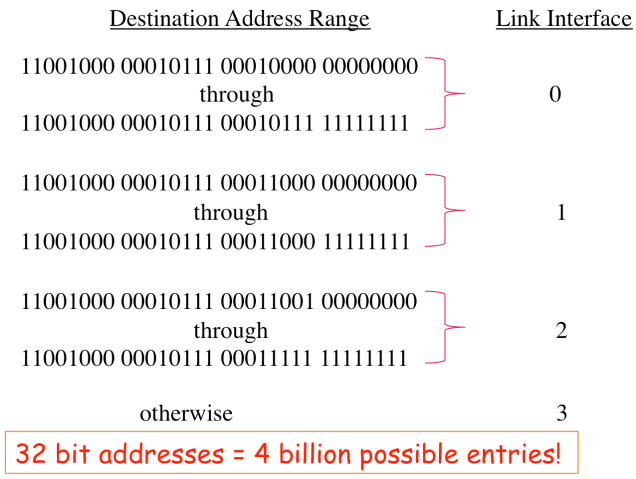


Figure 5.17: Example (impractical) forwarding table with one entry per IP address. (Figure by Kurose and Ross)

is allocated a (larger) block of IP addresses by IANA (of ICANN). An example is shown in Figure 5.18.

ISP's block	11001000 00010111 00010000	00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000	00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010	00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100	00000000	200.23.20.0/23
...	.....	....	....
Organization 7	11001000 00010111 00011110	00000000	200.23.30.0/23

Figure 5.18: An ISP's address space allocated evenly to eight customer organizations. (Figure by Kurose and Ross)

Instead of employing the very inefficient scheme of one entry per destination IP (giant forwarding tables), routing in the Internet is solved within the addressing hierarchy of the Internet as a network of (sub)networks. Routers do **longest prefix matching** for maintaining their forwarding tables and realizing their forwarding behavior as shown in Figure 5.19.

Groups of subnetworks called **autonomous systems** (AS) have freedom in how they deal with routing internally and they advertise to other autonomous systems the IP prefixes that are reachable through them as well as the associated cost (e.g. in number of subnets till the destination network etc). This is exemplified in Figure 5.20.

Now consider the situation after ISPs-R-Us (an autonomous system) has found a better route to Organization 1[3]. Figure 5.21 shows the new route advertisements in this case.

[3]For example, it may be that Organization 1's link connection to Fly-By-Night-ISP is somehow disconnected.

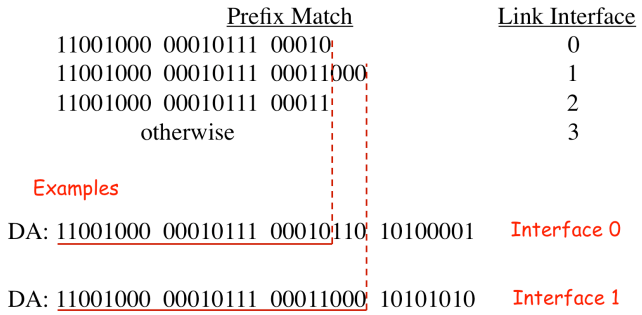


Figure 5.19: Example forwarding table with longest prefix match. (Figure by Kurose and Ross)

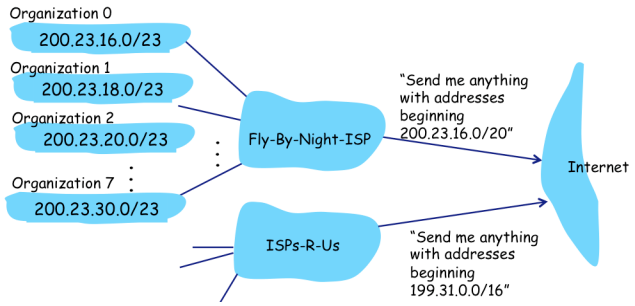


Figure 5.20: Routing advertisements. (Figure by Kurose and Ross)

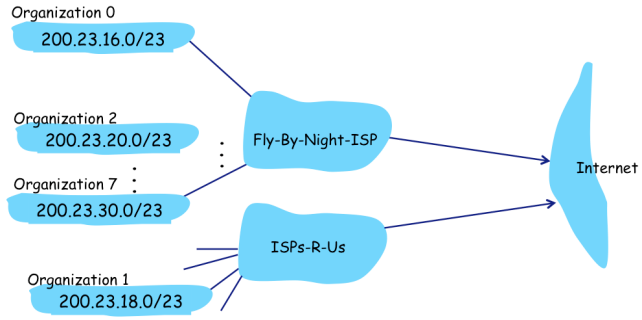


Figure 5.21: New routing advertisements after ISPs-R-Us has found a better route to Organization 1. (Figure by Kurose and Ross)

Internally, each autonomous system can run its own (intra-AS) routing protocol for reaching hosts inside the autonomous system. On the other hand, the (inter-AS) routing protocol at autonomous system boundaries (at border gateway routers) towards other autonomous systems must be standardized.

- Intra-AS routing protocols of the Internet:
  - **Routing Information Protocol (RIP)** - public: RFC 2453.

  - **Open Shortest Path First (OSPF)** - public: RFC 5340.
  - **Interior Gateway Routing Protocol (IGRP)** - proprietary.

- 'The' inter-AS routing protocol of the Internet:

  - **Border Gateway Protocol (BGP)** - public: RFC 4271.

Notice: The lecture slides contain sufficient explanations that are needed for the Internet's routing protocols, which are not repeated in this document (at least not this year: we are doing our best to produce text and enrich the lecture notes every year). It may be a good time now to switch to lecture slides, study the relevant slides and come back. Note that you are responsible for both the slides and these notes in the exam.

5.5 Network Layer Connections: Virtual Circuits

Packet-switched networks that provide a connection-oriented service, such as Asynchronous Transfer Mode (ATM) and Frame Relay, are called **virtual circuit** networks.

A virtual circuit consists of a path from source to destination where resources may be reserved (see Figure 5.22), virtual circuit identifier numbers (a different number for each link along the path) and the corresponding entries in router forwarding tables along the path. Link and router resources (e.g. priority in router queues, link bandwidth) may be allocated to a virtual circuit, resulting in predictable performance. As a result, the performance of a virtual circuit (the source-to-destination path) becomes similar to the performance of physical (e.g. telephone) circuits. For example, virtual circuits can provide a certain level of network layer reliability and better network performance per flow (e.g. lower delay, better throughput) as a service.

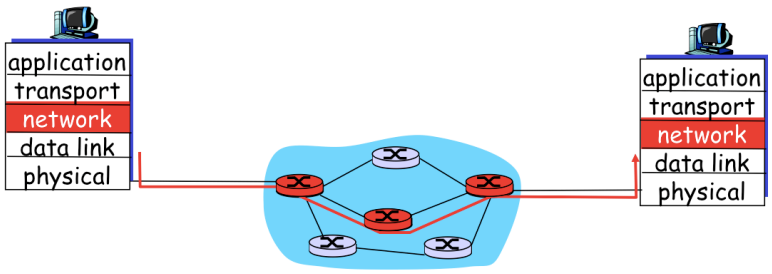


Figure 5.22: Virtual circuit: all packets taking the same path through the same routers from the source host till the destination host. (Figure by Kurose and Ross)

While traveling to their destinations, each packet indicates to the routers in the path which virtual circuit they belong to by means of a virtual circuit identifier (instead of a destination host address). For achieving the 'circuit-like' behavior, every router on the path maintains the states of the connections passing through. Remember that within a datagram network the same destination

address is processed by each router on the path. On the contrary, in virtual circuits the virtual circuit identifier can be a different number on each link. Each router on the host-to-host path maintains this information, i.e. the virtual circuit numbers in its incoming and outgoing links for each virtual circuit, as part of the connection state.

Similar to physical circuit switching, virtual circuit sessions (or calls) require call setup, call maintenance and call termination procedures. During call setup the caller initiates the call and the callee accepts the call by sending a reply as shown in Figure 5.23. After this point, the connection is established and data can start flowing in the virtual circuit.

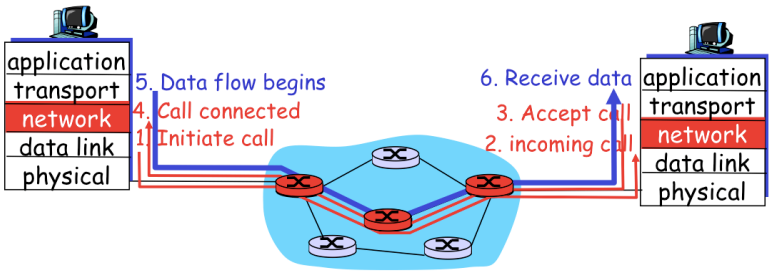


Figure 5.23: Virtual circuit call setup. (Figure by Kurose and Ross)

5.6 Summary

Network layer services are essential for host-to-host communication in computer networks. An important service is 'hierarchical' network layer addressing, which reduces routing complexity and makes it possible to deal with billions of IP addresses. Autonomous systems consisting of non-overlapping sets of subnets have different strategies for handling internal routing (intra-AS routing). In this way, each autonomous system can implement their own internal routing policies and optimize routing performance considering their own network topology. On the other hand, a common routing protocol is needed at the borders of autonomous systems for interoperability. The Border Gateway Protocol is the global standard of the Internet for routing packets across autonomous systems (inter-AS routing).

5.6.1 Literature

Suggested reading

- Chapter 4 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 4].

5.7 Homework Exercises

1. The address block starting with 190.100.0.0/16 is assigned to an ISP.

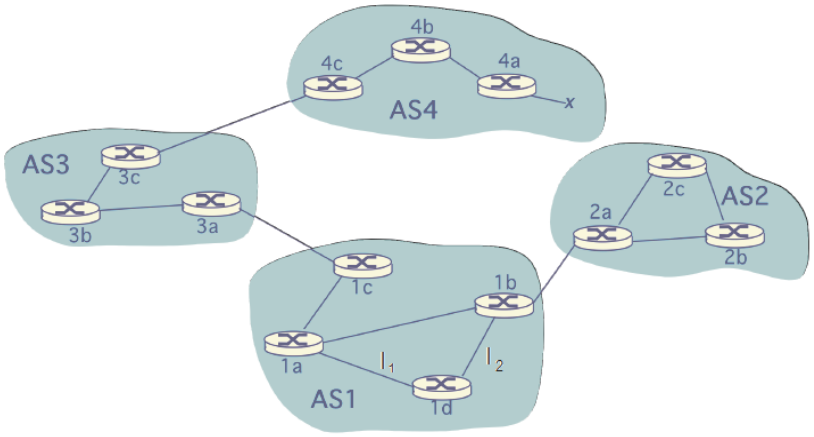
   (a) How many IP addresses are available to this ISP?

(b) The ISP needs to distribute these addresses to 3 groups of customers:

- 1st group: 64 customers; each needs 256 addresses
- 2nd group: 128 customers; each needs 128 addresses
- 3rd group: 128 customers; each needs 64 addresses

Design the sub-blocks and find out how many addresses are still available after these allocations.

2. Consider the network shown below, which consists of 4 autonomous systems.



Suppose AS3 and AS2 are running OSPF for their intra-AS routing protocol, and AS1 and AS4 are running RIP for their intra-AS routing protocol. Suppose eBGP and iBGP are used for the inter-AS routing protocol.

(a) Router 3c learns about prefix x from which routing protocol?

(b) Router 3a learns about x from which routing protocol?

(c) Router 1c learns about x from which routing protocol?

(d) Router 1d learns about x from which routing protocol?

(e) Router 4b learns about x from which routing protocol?

(f) Once router 1d learns about x, it will put an entry (x, l) in its forwarding table. Will l be equal to l_1 or l_2 for this entry? Explain in one sentence.

(g) Now suppose that there is a physical link between AS2 and AS4 (between 2c and 4a). Suppose router 1d learns that x is accessible via AS2 as well as AS3. Will l be set to l_1 or l_2? Explain in one sentence.

(h) Now suppose that there is another AS, called AS5, which lies on the path between AS2 and AS4 (not shown in diagram). Suppose router 1d learns that x is accessible via AS2 AS5 AS4 as well as via AS3 AS4. Will l be set to l_1 or l_2? Explain in one sentence.

3. What is the difference between routing and forwarding? Please explain.

4. A router has just received the following new IP address ranges: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21, 57.6.120.0/21. If all of these IP's refer to the same outgoing line interface, can they be aggregated? If so, to what? If not, why not?

5. What is the reason for having two types of routing algorithms working together (inter-AS and intra-AS protocols) for setting forwarding tables in the Internet?

6. Consider a datagram network using 8-bit host addresses. Suppose a router uses longest prefix matching and has the following forwarding table. For each of the four interfaces, give the associated range of destination host addresses and the number of addresses in the range.

Prefix Match	Interface
00	0
010	1
011	2
10	2
11	3

Chapter 6

Data Link Layer

In this chapter, we try to answer the following questions:

- What are the principles of link layer services and protocols?
- How can we address link interfaces of network devices?
- How can we detect packet errors at the link level?
- How can we control access to the link?
- What are the roles and working principles of hubs and switches?

The main responsibility of a data link layer protocol is node-to-node data delivery on a single link (in other words, 'one hop' data delivery). In doing this, the link layer has to deal with link layer packetization (framing), flow control, error control, addressing and Media Access Control (MAC). Packetization refers to the encapsulation of network layer datagrams inside link layer frames, including formatting of frame headers (prefix before data as usual) and trailers (suffix bit pattern). Flow control is again about speed matching between a sender and a receiver, but this time on a single link. Error control by the link layer protocol adds to the reliability of the link and is especially useful in links that are likely to introduce errors (e.g. in wireless). A multiple access protocol regulates how different nodes sharing the same link can gain transmission access to the link. **MAC addresses** are used in frame headers to identify the source and the destination nodes on a single link. Because of this the data link layer is also sometimes referred to as the **MAC layer**. The link layer uses the physical layer service: put bits on the medium. It is between the network layer and the physical layer as shown in Figure 6.1.

6.1 One Hop Data Delivery

A main responsibility of the link layer protocol is to make sure that the data is delivered to the desired one-hop neighbor node on the same link. There are two types of links. The first type is a **point-to-point link** where there are two nodes (one sender and one receiver), each connected to one end of the link (e.g. two ends of a wire).

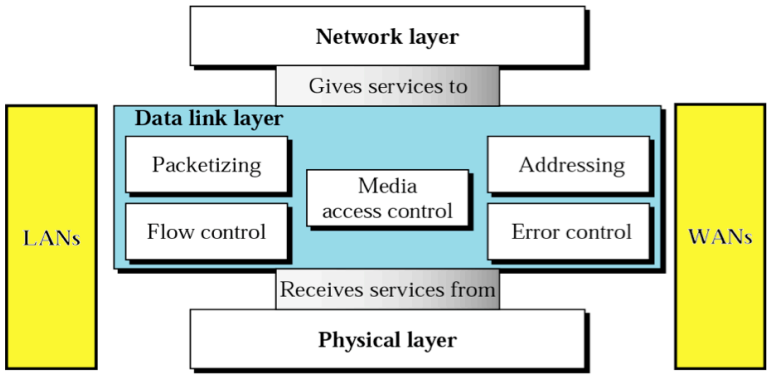


Figure 6.1: Issues addressed by the data link layer. (Figure by Forouzan)

The second type is a **broadcast link** where the network interfaces of many nodes (multiple senders, multiple receivers) are connected to a single shared broadcast medium (broadcast channel). There are two famous link topologies of a broadcast link: **bus topology** and **star topology**.

Bus Topology: In the past, the *bus topology* shown in Figure 6.2 was very commonly used in local area networks (LAN). In this topology, the data transmitted from individual nodes can 'collide' with each other since each node 'hears' every transmission in broadcast links. This is why, a bus topology link is said to have a single **collision domain**. This topology is nowadays more often used in networks of embedded devices and in the automotive industry (e.g. a vehicle's steering system communicating to the front wheels over the vehicle bus).

Definition 6.1.0.1 *Collision*

A collision on a network link refers to the situation where multiple simultaneous transmissions (overlapping is enough) on the same link interfere with each other at a receiver (or at multiple receivers) such that none of the transmissions can be correctly received by the receiver(s).

Star Topology: Today, the *star topology* shown in Figure 6.3 is very popular as the link layer topology in computer networks. In the star topology, there is an active entity sitting in the center, i.e. a *hub* or a *switch*, which has a single hop network interface connection to each node. In case of using a switch, simultaneous transmissions of different nodes are isolated from each other by the switch via buffering. In case of using a hub in the center, the star topology also has a single collision domain as hubs are nothing but dumb bit repeaters. We will discuss switches and hubs in more detail in Section 6.2, after we have introduced link layer addressing.

Remember the illustration of the comparison between the data delivery services provided by the different protocol layers (depicted again in Figure 6.4 for convenience). In the end-to-end path of a datagram between the source host

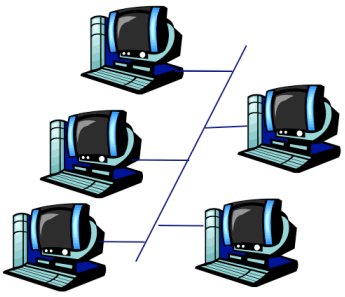


Figure 6.2: Bus topology. (Figure by Kurose and Ross)

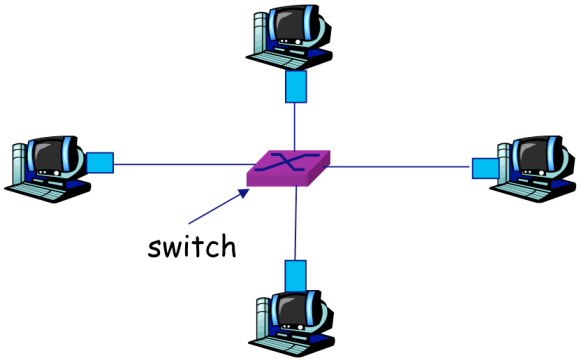


Figure 6.3: Star topology. (Figure by Kurose and Ross)

and the destination host, there can be various links of different types and with different link layer protocols. As a result, the quality of service and even the service types differ per link. For example, some links may be reliable while others are not. For *reliable* one hop data delivery, the link layer protocol needs to deal with duplicate frame receptions and data errors, while making sure that the data frames are delivered to the receiver in the same order in which they are supplied by the sender. Similarly, some links may provide a connection-oriented service while others provide connection-less service.

6.2 Link Layer Addressing

The link layer needs an addressing mechanism to be able to distinguish between nodes sharing the medium. For this purpose, it is common to employ 48-bit (6-byte) link layer addresses, which are represented using the hexadecimal notation (e.g. **0E-E3-87-BB-EB-B8**). In different contexts, a link layer address is sometimes called a *physical address* or *MAC address*. It is also possible that you see around technology specific names for the same; e.g. *Ethernet address*, *LAN address*. An example is shown in Figure 6.5.

The network interface cards of a node come from the manufacturer with their 'unique' link layer addresses assigned and 'hard-coded'. Therefore, link

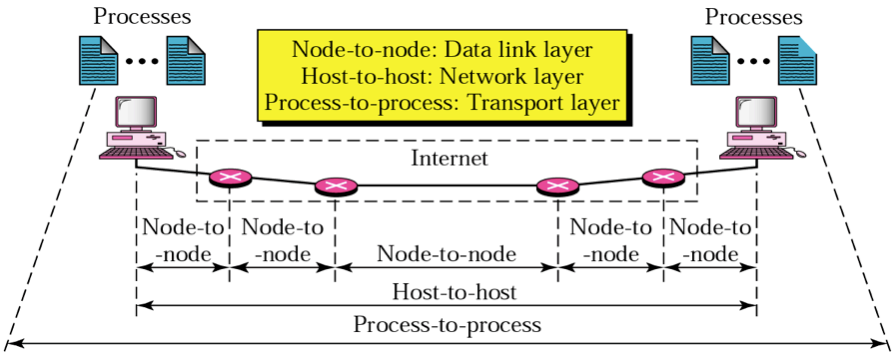


Figure 6.4: Connections at the transport layer (process-to-process), the network layer (host-to-host) and the link layer (node-to-node) shown in one picture. (Figure by Forouzan)

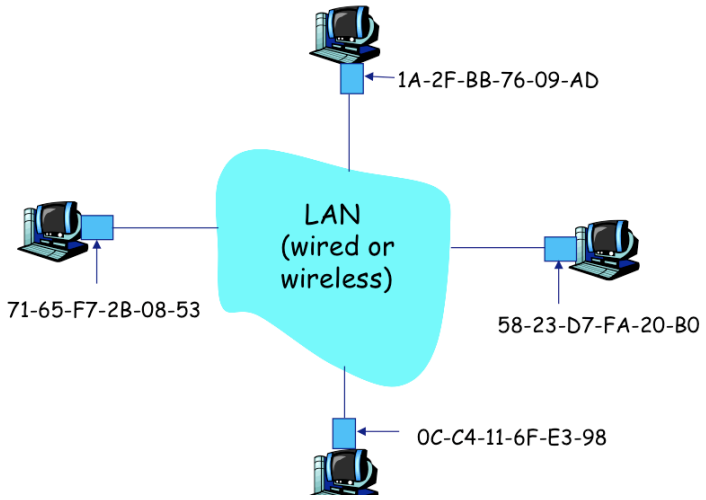


Figure 6.5: Each network interface has a unique link layer address. (Figure by Kurose and Ross)

layer addresses are considered to be static. For example, your laptop has the same MAC address whether you are connected to your home network or to the university network[1]. In practice, the link layer address of a network interface card can be changed in software.

Consider the delivery of a packet in the last hop before the destination host with IP address 137.196.7.78 as depicted by Figure 6.6. A service that maps the destination address 137.196.7.78 to the MAC address 1A-2F-BB-76-09-AD is needed[2]. In the Internet, this service (i.e. address resolution for routing in the same subnet) is provided by the **Address Resolution Protocol (ARP)**,

[1]Note that this is different than IP addresses, where you have a different IP address in each subnet. The first 24 bits of assigned to a manufacturers by IEEE. The remaining 24 bits are assigned by the manufacturer to its individual products' network interface cards.

[2]Note that every IP address is mapped to a (unique) MAC address.

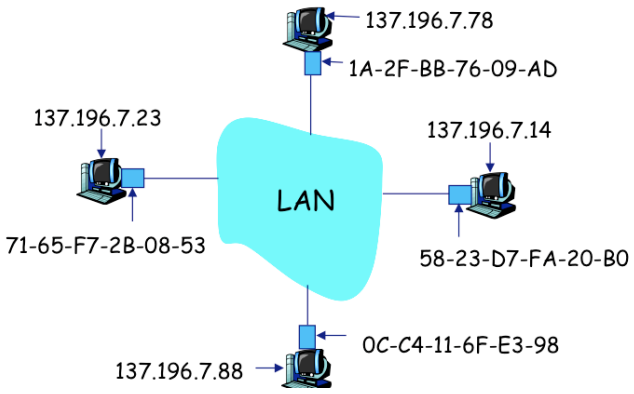


Figure 6.6: On a given link, every IP address maps to a MAC address. ARP does the translation from IP addresses to MAC addresses. (Figure by Kurose and Ross)

which is described in Figure 6.7. Every node maintains an ARP table, which contains mappings from some of the IP addresses on the same subnet to their MAC addresses, together with a TTL (Time To Live) value. TTL determines the time until the mapping expires (i.e. taken out from the ARP table) and is typically set to 20 minutes.

- A wants to send datagram to B, and B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF
  - all machines on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed

Figure 6.7: ARP in a nutshell. (Figure by Kurose and Ross)

Note that the creation of ARP tables does not require any manual support from a network administrator. The ARP table of a node is created fully automatically after joining a subnet.

6.3 Switches and Hubs

In a star topology, each host has a direct connection to a central node: a hub or a switch. In the following subsections we explain the operations of hubs and switches, emphasizing their differences.

6.3.1 Hub

A hub is simply a repeater (indeed a physical layer device), whose task is to forward and repeat bits coming into one of its link interfaces to all other links connected to its remaining link interfaces. A hub works on bits, it does not (necessarily) contain software and it does not examine frame headers. A hub does not buffer any frames and, therefore, it does not decouple transmissions of different senders. That means collisions can still occur in a star topology with a hub in the center.

6.3.2 Switch

As opposed to hubs, a switch[3] operates link layer frames, i.e. it examines frame headers of incoming frames. Therefore, their operation is much more complex than that of hubs. Switches are very often used in Ethernet links. A switch buffers the incoming frame, and depending on the destination MAC address it selectively forwards the frame on one or more of its link interfaces (i.e. store-and-forward behavior).

A switch maintains a **switch table** in order to keep track of which MAC addresses are reachable on each of its link interfaces. Every time the switch receives a frame on one of its link interfaces the switch adds an entry to its switch table mapping the source (link layer) address of the incoming frame to the corresponding link interface[4]. The switch operation is summarized in Figure 6.8.

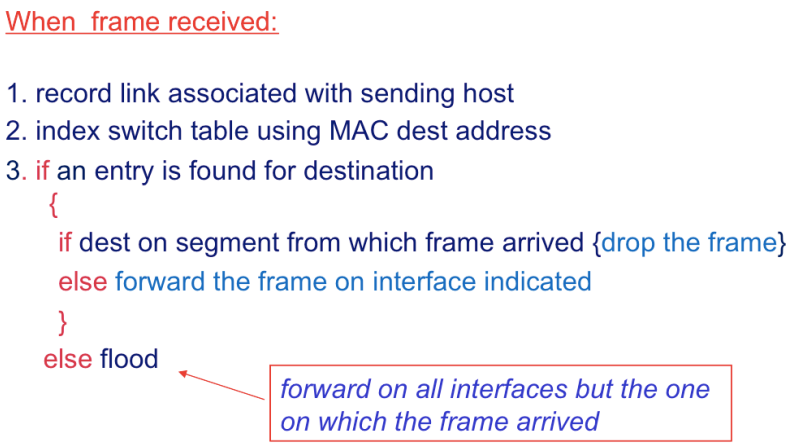


Figure 6.8: The operation of a switch in a nutshell. (Figure by Kurose and Ross)

There are three very nice properties of a switch.

[3]Not to be confused with layer 3 switches that operate on IP headers in hardware. The main difference of a router from a layer 3 switch is that a router's operation is in software and more dynamic, while a layer 3 switch's operation is in hardware and faster as a result. Layer 3 switches are beyond our scope.

[4]That is unless the entry already exists. Similar to the ARP table, the switch also maintains a TTL duration with each entry, at the end of which the corresponding entry is removed from the switch table.

- **Separate Collision Domains:** A switch employs the Ethernet protocol (discussed in Section 6.5) on each link separately. In this way, it creates a different collision domain on each of its links.
- **Transparency:** A switch is transparent to other nodes in its subnet.
- **Plug-and-Play Operation:** Switches are self-learning and they do not need to be configured by an administrator.

Switches can be interconnected as shown in Figure 6.9 and the switch operation depicted in Figure 6.8 and the build-up of switch tables remain exactly the same. See how the switch tables are built in this case in the video recording of the lecture (slides contain an example with animations).

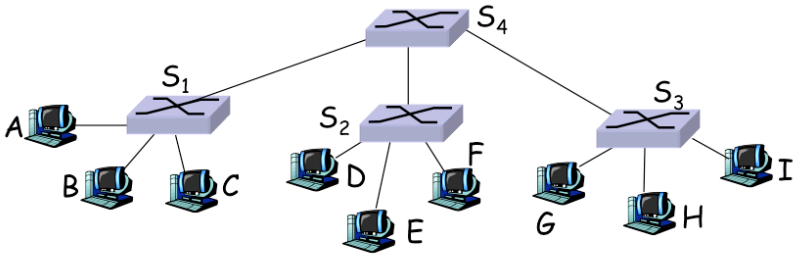


Figure 6.9: Multiple switches can be connected together in a subnet. (Figure by Kurose and Ross)

6.4 Link Layer Error Control

Bit errors can occur in links in the form of bit inversions, bit deletions and bit additions, e.g. due to interference, noise, reflections (wireless) and signal attenuation. Even though errors can be dealt with at higher layers, this may not be preferable especially considering heterogeneity of links. A single link with a high error rate on the end-to-end path can be the cause of dropping a lot of packets by the routers on that specific link (e.g. IP checksum failure). Therefore, detection and, if possible, correction of errors at the link level for error-prone links is advantageous.

For this purpose, the sender of a link computes error detection and correction (EDC) bits and adds this (redundant) information to the link layer frame header. The receiver (the other end of the link) checks for errors by doing the same computation after frame reception and comparing the result to the value in the frame header. Upon error detection, the receiver may choose to discard the frame or signal the sender to ask for a retransmission. Note that such error detection may not be completely reliable.

6.4.1 Bit Parity Checking

An example of adding redundancy EDC bits is used by **parity bit checking** as shown in Figure 6.10 where two versions of parity checking are considered:

single bit parity and two-dimensional bit parity. Single bit parity checking can detect all single bit errors. The parity bit aims to make the total number of 1's in the bit pattern even (even bit parity check) or odd (odd bit parity check). Two-dimensional bit parity checking structures the data into rows and columns and computes for each row and column the parity bit as it is done in single bit parity. Two-dimensional bit parity checking can detect and correct all single bit errors. The ability to correct is thanks to the fact that parity errors will be detected both at the row parity and the column parity for the erroneous bit. The only thing that is needed for correction is to flip the bit back to its original state.

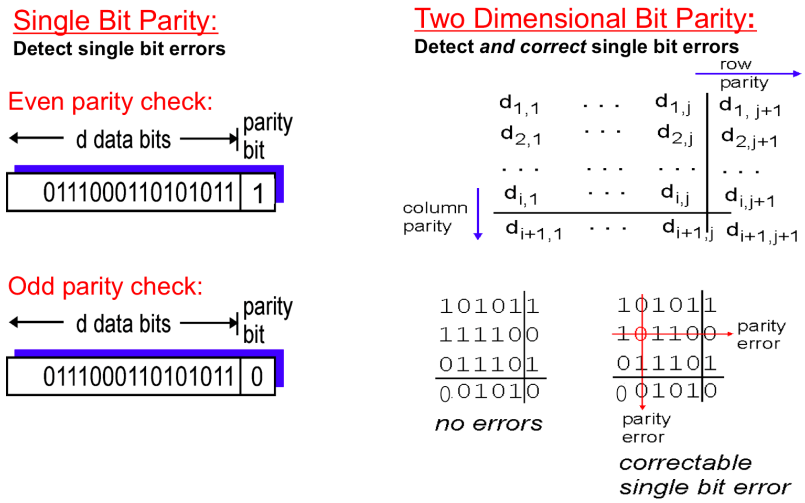


Figure 6.10: Parity bit checking. (Figure by Kurose and Ross)

6.4.2 Cyclic Redundancy Check

Another form of error checking that is widely used in practice is the **Cyclic Redundancy Check (CRC)**. In CRC, the data bits for error checking are considered to be a binary number D. A *generator* bit pattern G of length $r+1$ bits is used to generate a CRC bit pattern R of length r bits to be included in the frame header. R is the remainder of the division $\frac{D\times 2^r}{G}$ in modulo-2 arithmetic (without carries in addition and borrows in subtraction). The generated CRC bit pattern R satisfies the following condition: $< D, R >$ (i.e. the binary number formed by concatenating D and R) is exactly divisible by G in modulo-2 arithmetic (without carries in addition and borrows in subtraction). The receiving side uses the same generator bit pattern G to compute the division $\frac{<D,R>}{G}$. If the remainder of this division is zero, then the frame passes the CRC and its content is delivered to the layer above. If not, an error is detected. While parity bit checking performs very badly for bursty bit errors, CRC can detect bursts of length less than or equal to r. CRC bit pattern computation is shown in Figure 6.11 with an example.

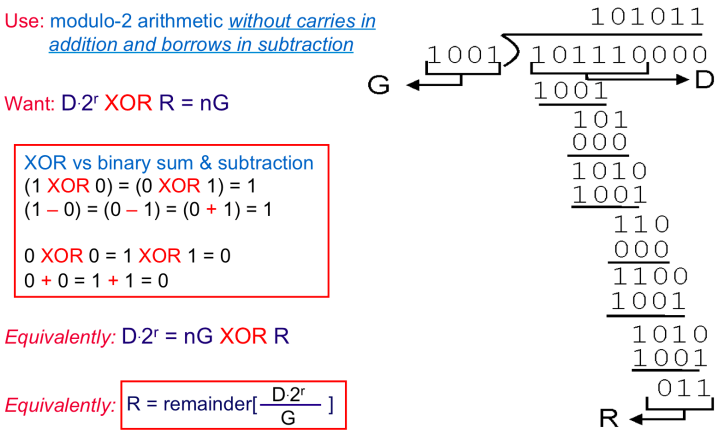


Figure 6.11: CRC bit pattern computation and an example. (Figure by Kurose and Ross)

6.5 Media Access Control

Media access control is everywhere. You can even observe it in how we do our lectures. A given classroom (e.g. Auditorium 3) is allocated to different lecturers at different time slots of a working week. The medium that is being shared is the physical space bounded by the lecture room's walls. Transmissions are often done by the lecturers, i.e. they do most of the talking, while students spend most of their time listening and taking notes. Of course, students can explicitly ask for 'medium access' by raising their hands at any time during a lecture. From time to time the lecturer inquires if there are students who want 'medium access' (e.g. asks if there are any questions). In this way, we can effectively communicate in lectures and maximize 'knowledge transfer'.

In this analogy:

- The classroom corresponds to the shared medium.
- The lecturer and the students correspond to the nodes.
- Media access control is through
  - reservation of the classroom for the lecture,
  - reservation of a 'question time' by students by raising their hands,
  - polling for questions by the lecturer,
  - and more mechanisms mainly stemming from our 'social protocol', e.g. do not interrupt others when they are speaking, give others opportunity to speak.

Now imagine listening to two lecturers at the same time in the same classroom. That would obviously be a disaster since none of the two lectures would be intelligible to the audience. As a result, students would learn nothing and very valuable resources, namely, the classroom time, the students' time and energy, as well as the lecturers' time and energy would be wasted. Clearly, media access control is crucial.

Similarly, frame collisions on a link waste channel and node resources and ideally they should be avoided altogether. In any case, the access to shared media must be regulated in order to prevent the situation that frame collisions constitute the majority of the transmission time into the medium. The protocols that regulate access to shared media are called **multiple-access protocols**. In the ideal situation, a multiple access protocol should divide the channel capacity **entirely** (no idle capacity) and **equally** among senders.

There are three classes of multiple-access protocols in the literature:

- Channel partitioning (channelization) protocols
- Random access protocols
- Controlled access protocols (taking turns)

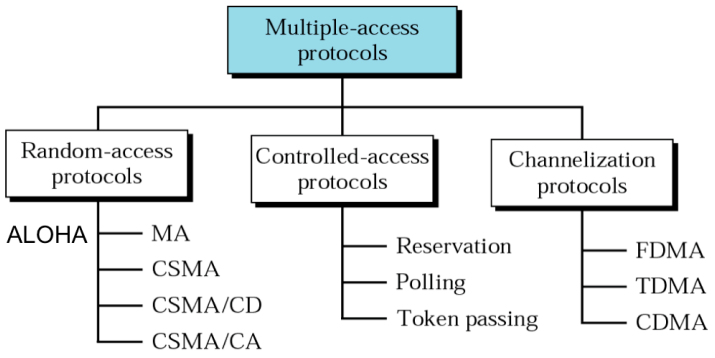


Figure 6.12: Three classes of multiple access protocols. (Figure by Forouzan)

6.5.1 Channel Partitioning (Channelization) Protocols

Channel partitioning means physically dividing the channel resource into pieces such that each transmission has its own 'dedicated' piece (i.e. no collisions). The partitioning can, for example, be done in the time dimension by working with time slots, in the frequency dimension by working with frequency sub-bands, and in the code dimension by working with signals that are orthogonal to each other[5]. Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and Code Division Multiple Access (CDMA) are the media access protocols used for these, respectively.

TDMA: Time is divided into periodic rounds. At each round a sender has the right to access the link only inside its allocated fixed-length time slot. The time slot length is typically chosen according to the transmission time of the maximum size packet (MTU). If a sender does not use its time slot, the slot is wasted. An example with 6 senders is shown in Figure 6.13.

[5]Two signals $f(t)$ and $g(t)$ are mutually orthogonal to each other in the interval $[0, T]$ if $\int_0^T f(t)g(t)dt = 0$.

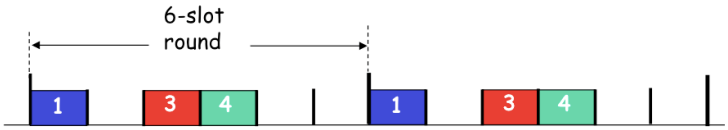


Figure 6.13: TDMA example. (Figure by Kurose and Ross)

FDMA: Fixed pieces of the total channel bandwidth are allocated to individual senders. Whenever a sender does not send anything its frequency band capacity is wasted. An example with 6 senders is shown in Figure 6.14.

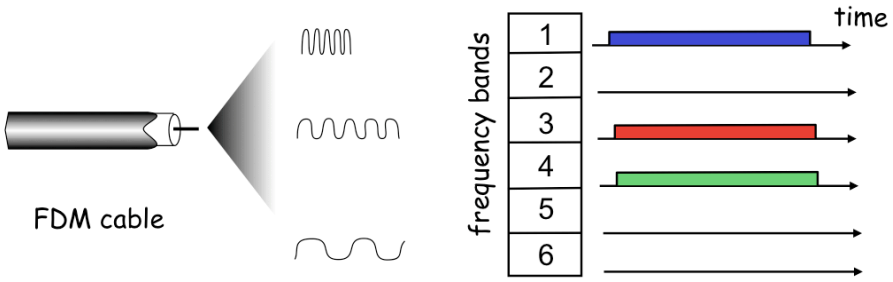


Figure 6.14: FDMA example. (Figure by Kurose and Ross)

CDMA: Each sender is assigned one of the mutually orthogonal codes, each code used to encode bit patterns. Multiple senders can transmit their bit patterns at the same time using the full channel bandwidth. The sum of the multiple signals of simultaneous senders ($s(t) = c_1(t)+c_2(t)+...+c_N(t)$) can easily be decoded at the receiving side by projecting the sum onto the individual codes. For example, projecting $s(t)$ onto $c_1(t)$ over one bit interval of length T seconds is by computing:

$$\begin{aligned}\int_T s(t) \cdot c_1(t)dt &= \int_T (c_1(t) + c_2(t) + ... + c_N(t)) \cdot c_1(t)dt \\ &= \int_T c_1(t) \cdot c_1(t)dt + ... + \int_T c_N(t) \cdot c_1(t)dt \\ &= \int_T c_1(t) \cdot c_1(t)dt\end{aligned}$$

6.5.2 Random Access Protocols

As opposed to channel partitioning protocols, random access protocols do not physically divide the link resources among senders. As a result senders can randomly assume full channel resources when they are active. While this means collisions can happen, a random access protocol is designed to deal with those: detect collisions and recover. **ALOHA** protocol, **Carrier Sense Multiple Access (CSMA)** and their variants are examples of this class of multiple access

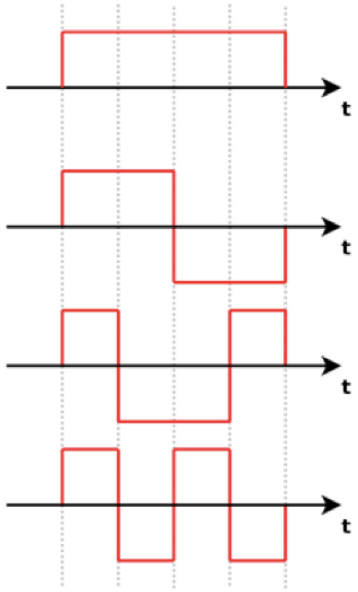


Figure 6.15: Four orthogonal codes, each of which can be used by a different sender. (Figure by Wikipedia)

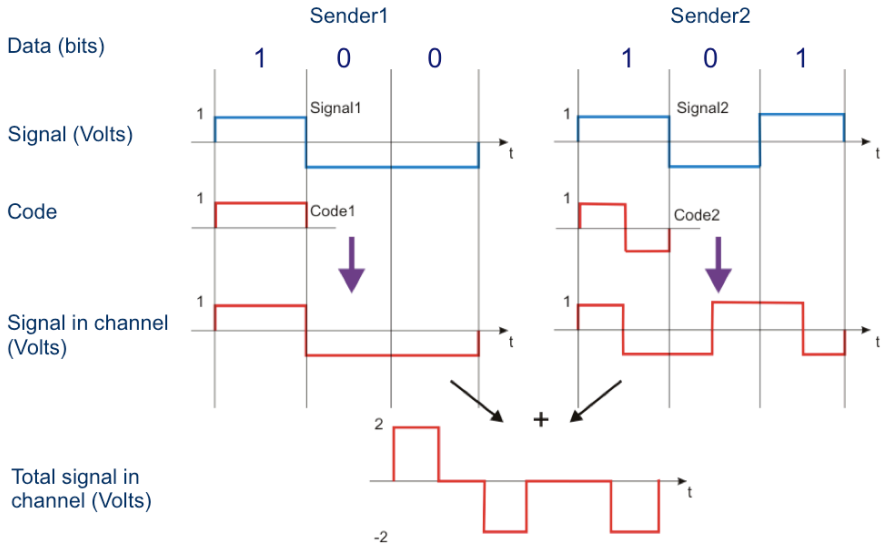


Figure 6.16: CDMA encoding with two senders and two codes. The bottom signal is $s(t)$. Note that for the first sender, the results of the computation $\int_T s(t) \cdot c_1(t)dt$ in the first, second and third bit intervals are 1, -1 and -1 Volts, corresponding to bits 1, 0 and 0 respectively. For the second sender, the results of the computation $\int_T s(t) \cdot c_2(t)dt$ in the first, second and third bit intervals are 1, -1 and 1 Volts, corresponding to bits 1, 0 and 1 respectively.

protocols. To avoid repetition, in the lecture notes we merely introduce these protocols briefly. See lecture slides for detailed explanations and diagrams.

ALOHA: The name was given by its designers at the University of Hawaii (a word used for greeting in Hawaiian). In ALOHA, a sender that has data to send immediately transmits a full link layer frame containing (part of) the data. In case there is a collision (packet loss), the sender retransmits the frame with the probability p after it has completely transmitted the collided frame (waste). On the plus side, this is a decentralized protocol, i.e. there is no central intelligence that decides who can transmit when. However, the efficiency (probability of a successful transmission attempt) of ALOHA is very bad, leading to a lot of retransmissions and poor performance. ALOHA has time slotted (needs synchronization among senders) and unslotted versions.

CSMA: Since frequent collisions kill efficiency as in the case of ALOHA, it was necessary to develop a protocol that aims to prevent collisions. CSMA is such a protocol "not interrupting others while they are transmitting". The CSMA sender listens to the channel before transmitting anything (carrier sensing). If there is a currently ongoing transmission, the sender defers its transmission. The operation of the CSMA sender is given in Figure 6.17.

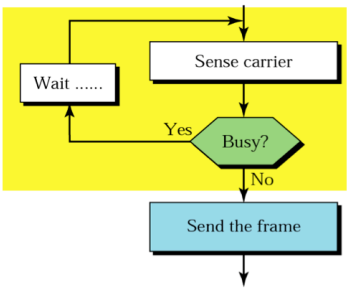


Figure 6.17: CSMA sender operation. (Figure by Forouzan)

Note that collisions can still occur with CSMA due to propagation delays as shown in Figure 6.18. A variant of CSMA, **CSMA with Collision Detection (CSMA/CD)** tries to detect collisions and quickly stops the sender's transmission upon collision detection (less resources wasted). The sender can detect a collision, for example, if it is receiving another signal while sending its own. There are many ways to detect collisions. A popular MAC protocol that employs CSMA/CD is the Ethernet protocol.

Another variant of CSMA, **CSMA with Collision Avoidance (CSMA/CA)** does not employ collision detection. CSMA/CA proves to be useful especially in wireless networks where collision detection is very difficult. There are two difficulties with collision detection in wireless. The first difficulty is that the power of the received signal is usually very weak in comparison to the power of the transmitted signal[6]. Secondly, the so

[6]This is as measured by the transmitter. The weakness of the received signal comes from factors such as path loss, reflection and interference.

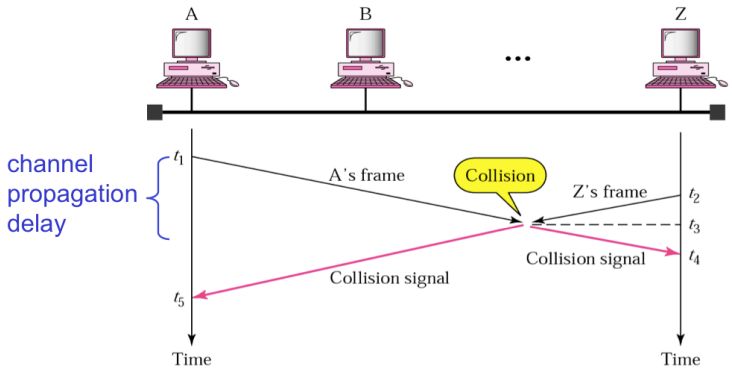


Figure 6.18: Collisions are not completely eliminated by CSMA. (Figure by Forouzan)

called **hidden terminal** problem depicted in Figure 6.19 can cause packet collisions that are impossible to detect by the transmitters.

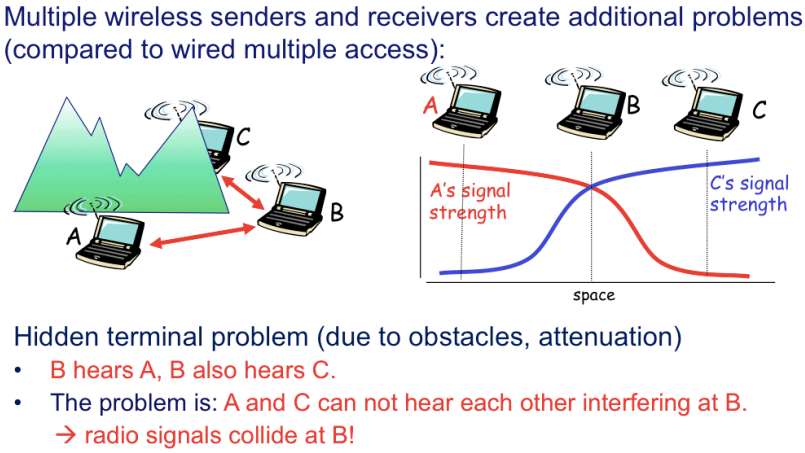


Figure 6.19: The hidden terminal problem. (Figure by Kurose and Ross)

If the channel is sensed busy, the CSMA/CA sender waits for a randomly chosen amount of time before it tries again, which is called a 'random backoff'. When the backoff counter hits zero (i.e. the randomly chosen backoff period expires), the sender 'senses' the channel again and transmits its packet if the channel is not busy. If not, a new random backoff duration is chosen (from a larger interval) and the sender waits again. This is repeated until the receiver's acknowledgement is received for the frame. The CSMA/CA operation of a WiFi link (IEEE 802.11 protocol) is shown in Figure 6.20.

Additionally, the CSMA/CA may allow (wireless) senders to reserve the channel (instead of random access) with the purpose of eliminating collisions. For this, the CSMA/CA sender transmits small Request-To-Send (RTS) message to the wireless access point, which in return broadcasts a

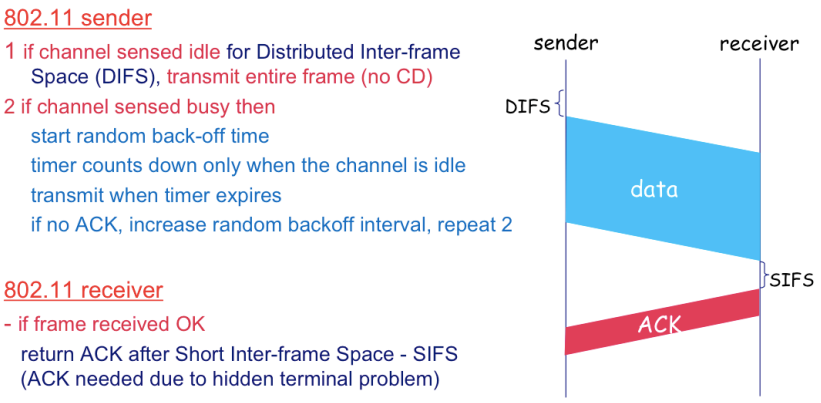


Figure 6.20: IEEE 802.11 (WiFi) CSMA/CA in a nutshell. (Figure by Kurose and Ross)

Clear-To-Send (CTS) message. The CTS message indicates which node is allowed to transmit next. Upon receiving the CTS broadcast, the node that is cleared for transmission will transmit its frame while the other nodes will wait. Note that RTS messages can still collide with each other and with CTS messages of the access point. However, since these are tiny packets the loss due to such collisions is minimal compared to the gain from reduced numbers of large frame collisions. The CSMA/CA operation with channel reservation (collision avoidance) is shown in Figure 6.21.

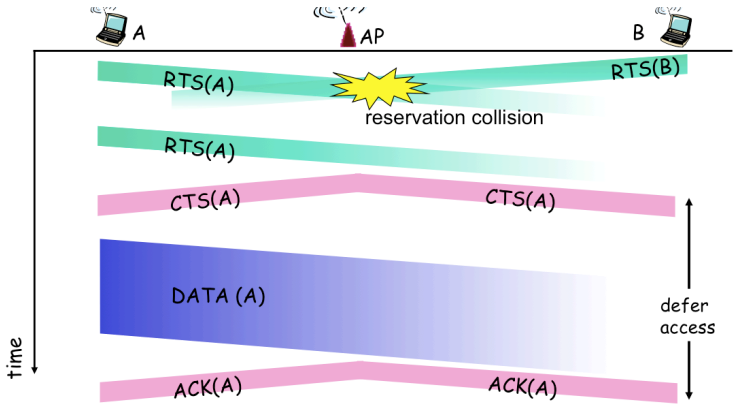


Figure 6.21: CSMA/CA operation with channel reservation. (Figure by Kurose and Ross)

6.5.3 Controlled-Access (Taking Turns) Protocols

The third class of multiple access protocols is the controlled-access protocols, where nodes take turns in accessing the channel. A main difference from TDMA is that taking turns protocols do not allocate a fixed channel time to individual senders. On the contrary, a sender's access time to the channel (the duration of

its turn) can be increased if the sender has more data to send than others. While idling of (rather long) time slots is a problem of TDMA, it is not a problem for controlled-access. We will mention two taking turns protocols: i) polling, and ii) token passing.

Polling: In polling, a 'master node' invites 'slave nodes' (transmitters) to access the channel in turns by sending them polling messages one by one. Although this protocol is very simple for the slave nodes to run, polling causes traffic overhead and introduces delays. Furthermore, if the master node fails then the link is not usable by any senders (single point of failure).

Token Passing: In token passing, a control token message is passed from node to node in a logical (token) ring. The node that currently has the token can transmit and pass the token to the next node after it's done. Similar to polling, the necessity to circulate the token message gives channel overhead and latency. This time the token message itself is the single point of failure since nodes will not transmit a token message if they have not received one (e.g. if the token packet is lost).

6.6 Summary

The link layer is responsible for transferring a packet from one node to adjacent node over a single link. Media access protocols govern which node on a given link is allowed to transmit at any given time. In doing so, these protocols should aim to utilize the channel optimally and fairly. We have seen three classes of media access protocols: i) channel partitioning (channelization) protocols, ii) random access protocols, and iii) controlled access protocols (taking turns). Wireless links bring unique challenges. The reader should refer to the lecture slides and the video recordings of the link layer lectures for a full coverage of MAC protocols and wireless link challenges. The lecture notes are focused on the fundamentals rather than the specific protocols. Note that TCP interprets transmission timeouts as indicators of congestion and decreases its congestion window size. Although this assumption is acceptable when the links are unlikely to lose packets, when we consider very lossy wireless links on the host-to-host path we can immediately argue that a lot of timeouts can occur due to losses on a wireless link. For example, you may try to take your laptop to the kitchen and download a file over WiFi while the microwave oven is on. You will see that your wireless connection will suffer considerably since microwave ovens operate at roughly the same frequency range with a your WiFi access point. The massive interference from the microwave oven will cause packet losses and timeouts, and these losses have nothing to do with packet congestion in the network core.

6.6.1 Literature

Suggested reading

- Chapter 5 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 5].

- Chapter 6 of the book Computer Networking: A Top-Down Approach by Kurose and Ross [20, Chapter 6].

6.7 Homework Exercises

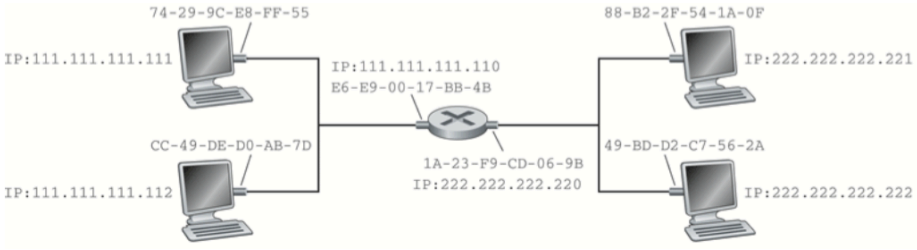
1. Why do we need both link layer and end-to-end (transport layer) reliability?

2. If all the links on the end-to-end path from the source host to the destination host are reliable, would that be (by itself, without using TCP) sufficient for the reliability needs of a file transfer application? Why?

3. Assume that the sender and the receiver on a link agree on a 4-bit generator pattern, G, which is equal to 1001. The sender wants to send to the receiver 6 bits of data, D, which is equal to 101110. Find the CRC bits R to be appended to D. *Show your work*

4. Suppose nodes A and B are on the same 10 Mbps broadcast channel, and the propagation delay between the two nodes is 325 bit times. Suppose CSMA/CD and Ethernet packets are used for this broadcast channel (see slides). Suppose node A begins transmitting a frame and, before it finishes, node B begins transmitting a frame. Can A finish transmitting before it detects that B has transmitted? Why or why not? If the answer is yes, then A incorrectly believes that its frame was successfully transmitted without a collision. ***Hint:*** Suppose at time t = 0 bits, A begins transmitting a frame. In the worst case, A transmits a minimum-sized frame of 512 + 64 bit times. That means A would finish transmitting the frame at t = 512 + 64 bit times. Thus, the answer is no, if B's signal reaches A before bit time t = 512 + 64 bits. In the worst case, when does B's signal reach A?

5. Consider the network depicted in the figure below. The IP addresses and MAC addresses of individual interfaces are as denoted in the figure.



Suppose that the sender host with the IP address 111.111.111.111 wants to send an IP datagram to the receiver host with IP address 222.222.222.222. Answer the following questions:

(a) How many subnets are there in this network? Which IP addresses belong to which subnet?

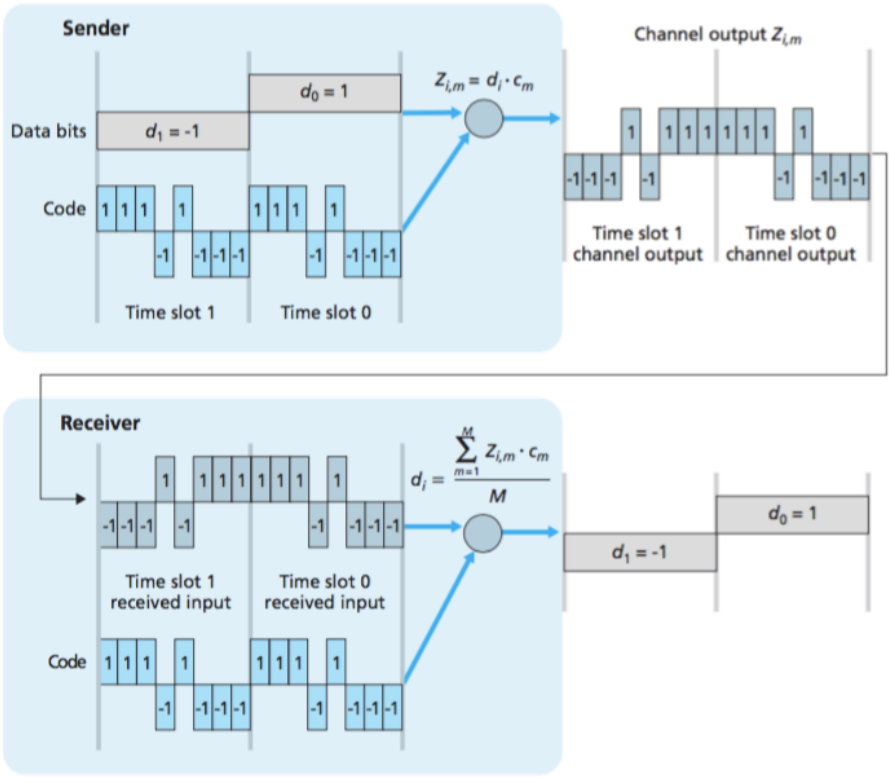
(b) What is the destination IP address of the datagram when it leaves the sender host? What is the destination IP address of the datagram when it leaves the router?

(c) What is the destination MAC address of the frame when it leaves the sender host?

(d) What is the destination MAC address of the frame when it leaves the router?

(e) Which protocol is used to determine the destination MAC address?

6. Consider the single-sender CDMA example in the following figure. What would be the sender's output (for the 2 data bits shown) if the sender's CDMA code were (1, -1, 1, -1, 1, -1, 1, -1)?



Chapter 7

Authentication and Authorization

Applications interacting with end users access resources that represent significant value (e.g. entering a building, reading or modifying a data base, performing computations) either directly or over a network. Many security goals of a network are thus related to access and proper use of these resources. As defined in Chapter 1, a *security policy* specifies:

- What are the security attributes that should be achieved?
- When should these security attributes achieved?
  - ... for which resources?
  - ... in which context?

> *An application security policy that we want is an access control policy; one that specifies 'proper use' in terms of* ***who can perform what actions on which resources and when****.*

Enforcing the desired access control policy requires checking that an attempted action on a resource is allowed for the user performing the action. Thus we have to perform **authentication**; check the identity of the user (or at least establish some properties of the user such as 'the user is a student') and **authorization**; check the actions by the user on the resource against the policy.

This chapter addresses the following questions:

- What are the components of access control?
- How to achieve authentication and authorization?

7.1 Access control

While the desired security policy describes allowed access, access control enforces such policy. A policy, expressed at a high level of abstraction, may be 'the

User \ Resource	Gradelist 2IC60	Submission-Program
Tanir	Read, Write	Update
Jerry	Read, Write	Update
Alice	Read	Execute
Bob	Read	Execute
Charlie	Read	Execute

Figure 7.1: An access control matrix with five users and two resources.

lecturer maintains an online gradelist that the students can view and an online essay submission program that the students can run'.

Recall from Chapter 1 that the meaning of a security policy is given by the interpretation in a security model. An obvious choice for the structure of the mathematical security model is a relation on subjects, resources and rights.

Our goal is to define *access control policies* which capture the meaning of the intended high level policy by somehow specifying who (**subject**) is trusted with which resource (**object**) to do what (**allowed actions**).

7.1.1 Access Control Matrix

A basic format in which one can give an access control policy is the *access control (AC) matrix*. In this matrix there is a row for each subject and a column for each resource. This results in a field for each combination of a subject and a resource. The rights that that the subject has on that resource, i.e. allowed actions, are filled in this field. The example in Figure 7.1 shows the access control matrix capturing the high level policy given above for access by the lecturers Tanir and Jerry, and the students Alice, Bob and Charlie to the course resources *gradelist* and *essay submission program*.

In going from a high level policy to an AC matrix we have to translate general operations (view, maintain, run) to specific actions on the system (read, write, update, execute) and we also have to instantiate general terms (lecturer, student, gradelist) to specific users/resources (Tanir, Jerry, Alice, Bob, Charlie and Gradelist 2IC60).

The AC matrix gives us an easy to understand way of specifying the access control policy; it is immediately clear from the AC matrix who has what rights on which resources. Yet it has several drawbacks. For one, it is quite far from the high level security policy: we have to do quite a bit of manual work to capture the policy in an AC matrix. Besides giving more work, the large gap also leaves additional room for mistakes and misinterpretations. It is difficult to make sure that the AC matrix is a 'good' implementation of the high level policy. There is no way of exactly defining when the AC matrix is 'good' without doing the translation by hand again.

Another drawback of the AC matrix is that a lot of the original policy is lost in this translation For example in Figure 7.1 we can see that Alice may only read but not that this is because she is a student. This also impacts maintainability of the AC matrix; recall that it is meant to capture the high level policy 'the lecturer maintains a gradelist that the students can read and an essay submission program that student can run'. Thus, if the students in

Role \ Resource	Gradelist 2IC60	Submission-Program
Lecturer	Read, Write	Update
Student	Read	Execute

Role	Users
Lecturer	Tanir, Jerry
Student	Alice, Bob, Charlie

Figure 7.2: An RBAC policy with two roles, two resources and five users.

the class change we need to update the matrix. If Alice leaves the class we may need to revoke her rights but how do we know that Alice had the read right (only) because she was a student in the class?

The matrix for an entire system/network is difficult to manage. A centrally stored matrix would create a huge bottleneck for the system/network as any action on any resource would need to be checked at this central point. Instead we need to distribute the storage of the matrix. The use of **Access Control Lists** is one way of doing this. An AC list is basically a column from an AC matrix, stating all the rights that different subjects have on a single resource. As such it has a natural place to store it, i.e. together with the resource. Of course a problem is if rights change (e.g. a student is added or removed from the class) all relevant AC lists have to be found and updated. While this gives a viable implementation, the maintainability only gets worse.

Finally, to use his read right Bob will have to identify himself to the system. This should not be needed; Bob should only have to prove that he is a student and not reveal who he is. Below we look at different mechanisms to address these issues with the access control matrix;

7.1.2 Role Based Access Control (RBAC)

The high level policy 'the lecturer maintains a gradelist that the students can read and an essay submission program that student can run' can be expressed in terms of rights that users within a certain role have. This is very common in authorization (access control) policies. It is an instance of the general principle that users should have the permissions needed to perform their tasks (and not more). The tasks of a user are determined by the role they play. As such it would be useful to describe access control in terms of roles which, as you may have guessed from the name, is what role based access control (RBAC) does.

Figure 7.2 shows a role based AC policy that tries to capture the high level policy. The resulting access rights are the same as before, i.e. if we construct an AC matrix for the RBAC policy in Figure 7.2, we get the one in Figure 7.1. Assigning rights to roles rather than users reduces the number of rows needed. We need only a single row for all students together instead of a row for each student. Of course we also need to keep track of who has what role, which results in an additional table containing role assignments to users.

At first glance the added level of indirection may seem to actually increase the amount of work to be done; in addition to assigning rights, now to roles rather than to users directly, we also need to assign roles to users. Yet, one advantage is already seen from comparing the example in the figures: with roles we assign rights in batches. For example, we assign both the rights 'Read on Gradelist 2IC60' and 'Execute on Submission-Program' by giving the role student to some user. More importantly, the RBAC policy is closer to the high

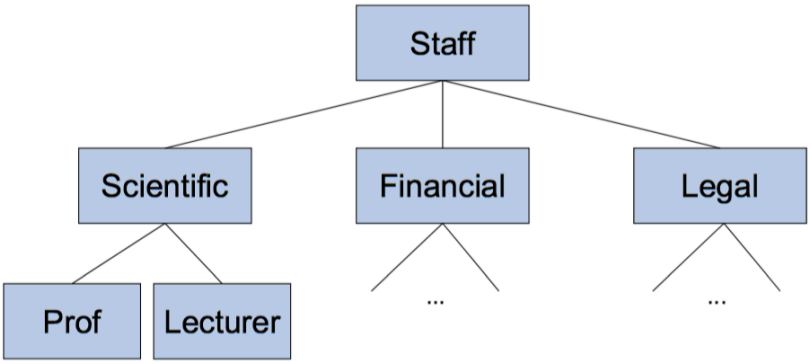


Figure 7.3: An example role hierarchy for a university.

level policy. It retains the notions of 'Lecturer' and 'Student' and the notion that having one of these roles is why users obtain certain permissions.

By retaining more of the intention of the high level policy RBAC helps improve maintainability. If Alice leaves the class all we have to change now is the role-subject table. We do not need to manually change any rights. When Alice tries to read Gradelist 2IC60 the system will compute her rights which, as she is no longer a student, do not include reading this gradelist. Thus access will be denied. If the high level policy changes and students are no longer allowed to submit essays (for example because the deadline has passed) only this entry in the role-resource table needs to be changed.

Hierarchical RBAC Roles can provide a huge improvement in maintainability. However, especially in large organizations, the number of roles can quickly grow to unmanageable proportions. Role hierarchies help bring structure to the roles. A university can define different categories of staff (see for example Figure 7.3), each with their own rights and responsibilities. By using a role hierarchy we only have to specify shared rights once; e.g. we may assign the right to supervise students to professors and the right to enter buildings to staff. As members of staff, the professors will inherit the rights from staff rights and as a result they will also be allowed to enter the building.

7.1.3 XACML for Attribute Based Access Control (ABAC)

A role is just an example of a property (or *attribute*) that we may want to link a right to. In order to capture the high level policy "Every student who has their Bachelor and has passed courses 2IC60 and 2IC70 may enroll for course 2IS25" we need to assign a right (enroll for 2IS25) to a combination of attributes ("is student", "has Bachelor", "passed 2IC60", "passed 2IC70"). Of course, we could make a role capturing exactly this combination of properties. However, we might need many of these roles and anytime anyone passes a course we have to adapt memberships of such roles. Obviously, it would be better to assign a right directly to combinations of properties.

We could do the same for resources to allow us to assign rights to a group of resources at the same time. For example, we may want to specify a policy such as 'students may read all files for course 2IC70'. For the action we can again do

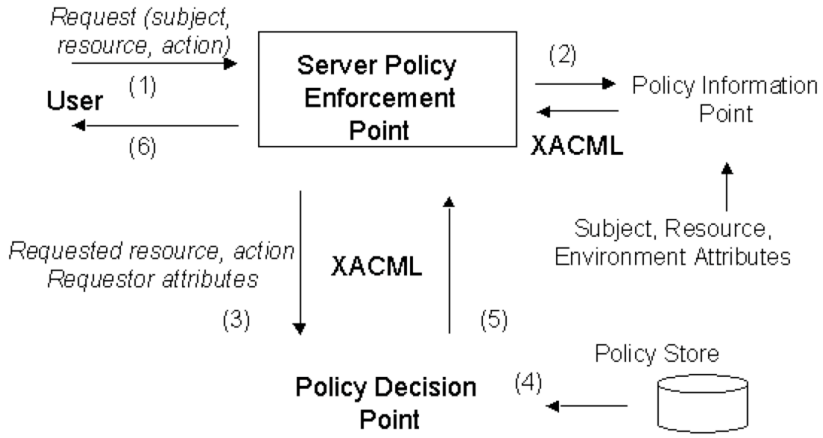


Figure 7.4: Simplified XACML request evaluation flow.

the same; instead of allowing action 'read' we could use attributes as in 'allow any action marked as *viewing*' to allow actions read, display, print, etc. Finally environmental conditions such as 'during working hours', 'before the deadline' can also be captured with attributes. By assigning rights to combinations of such attributes, attribute based access control (ABAC) provides a very powerful language for expressing access control policies.

The eXtensible Access Control Markup Language (XACML) standard defines a popular ABAC language and a system/network architecture for enforcement. XACML defines several components involved in the AC enforcement. Figure 7.4 gives a (simplified) view of these components and their interaction. The Policy Enforcement Point (PEP) is responsible for intercepting requests and ensuring that users only get access to their allowed resources. The PEP uses a Policy Decision Point (PDP) to determine which requests are allowed (should be granted). The PEP gathers information about the subject and the resource and the context in which the request is being issued (the environment) from a Policy Information Point (PIP) and incorporates this in a request sent to the PDP. The PDP tests the request against the set of policies that it has in its policy store to make a decision (access granted or access denied). The PDP can also return values indicating it is unable to make a decision: indeterminate (some error occurred) or not-applicable (this PDP has no policies related to the request).

A policy set contains, in addition to a list of policies[1], a combination algorithm that is used to combine the different decisions of the policies in the set. Examples are 'first applicable'; the first policy to return a decision gets selected, 'DENY overrides'; if a single policy causes to deny the request this is the end decision, even if others allow the request and 'PERMIT overrides' where a single permit decision overrides any other decisions.

A policy has a target determining which requests it is (or at least might be) applicable to. The rules of the policy will be evaluated for those requests. The rules check conditions (e.g. the issuer of the request is a student) to either PERMIT or DENY a request. The policy also has an algorithm to combine the

[1] A policy list would be a better name than a policy set since the order may indeed matter.

```
<Policy PolicyId="ExamplePolicy" RuleCombiningAlgId=
        "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
<Target>
 <Subjects> <AnySubject/> </Subjects>
 <Resources> <Resource>
  <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            >http://server.example.com/code/docs/developer-guide.html</AttributeValue>
   <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
  </ResourceMatch>
 </Resource></Resources>
 <Actions> <AnyAction/> </Actions>
</Target>
<Rule RuleId="ReadRule" Effect="Permit">
 <Target> <Subjects> <AnySubject/> </Subjects>
 <Resources> <AnyResource/> </Resources>
 <Actions> <Action>
  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
   <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
  </ActionMatch> </Action> </Actions>
 </Target>
 <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
   <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="group"/>
  </Apply>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
            >developers</AttributeValue>
 </Condition>
</Rule>
</Policy>
```

Figure 7.5: Part of an XACML policy.

answers of different rules. Finally obligations may be associated with a decision; e.g. permit the read operation but with some action that should be executed; e.g. notify the data owner, delete any copy of the data within a week, etc.

Policies are written in XML[2] making them human and machine readable. The example in Figure 7.5 shows part of a policy which allows a developer to read the developers guide document. The generality and power of XACML as well as the fact that it is too cumbersome to specify policies by hand using XACML are clear from this simple example. While writing a single policy is not always easy, when having to maintain a whole set of policies, possibly written by multiple parties, it can become a real challenge. Therefore, automated analysis tools for XACML (see for example [25, 26] and further links provided there) have been developed that help a user in maintaining policies. These tools check properties such as who has access to certain resources and the impact of a policy change (i.e. the effect of updating a certain policy).

The requests and responses are also formulated in XML. The request basically describes who (user and attributes; e.g. Seth from group developers)

[2]XML stands for Extensible Markup Language (XML) and it is a human readable data format. Though XML (and thus XACML) format is human readable it is not very human friendly. Interpreting a policy can be hindered by the large amount of textual overhead.

```
<Request>
 <Subject>
   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
    <AttributeValue>seth@users.example.com</AttributeValue>
   </Attribute>
   <Attribute AttributeId="group"DataType="http://www.w3.org/2001/XMLSchema#string"
     Issuer="admin@users.example.com"> <AttributeValue>developers</AttributeValue>
     </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
     DataType="http://www.w3.org/2001/XMLSchema#anyURI"> <AttributeValue>
     http://server.example.com/code/docs/developer-guide.html </AttributeValue>
     </Attribute>
 </Resource>
 <Action> <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
     DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>read</AttributeValue> </Attribute>
 </Action>
</Request>
```

```
<Response>
 <Result>
  <Decision>Permit</Decision>
  <Status>
   <StatusCode
   Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
  </Status>
 </Result>
</Response>
```

Figure 7.6: An XACML request and response (source: `sunxacml.sourceforge.net/guide.html`).

wants to do what (action; e.g. read) with which resource (e.g. developer guide). It may also contain information about the environment; e.g. the request was made from within the same office (or from within the same subnet) and during working hours. The response contains the decision and a status code.

The XACML system is very general and by using selected sets of attributes and formats it can model or be used in combination with other identification and access control systems and standards. Note that XACML is attribute based so we do not always need the identity of the user (requester, also called subject in XACML terms). However, as the identity is just another attribute (named 'urn:...:subject-id') we can use it in the policy. Example: Give read access to any one with a role attribute 'student' or an identity attribute 'Jerry' or an identity attribute 'Tanir'.

7.1.4 Distributed Access Control

Thus far we have considered a single authority based AC where all access rights, roles assignments etc. are specified and enforced by a single entity. However, on a computer network there are multiple parties each with their own resources and policies for use of these resources. One could consider each of these an authority maintaining an access control policy, for example a hierarchical RBAC policy, for their own resources. For example, TU/e may want to allow use of some of its resources by students of another Dutch universities. With RBAC this would require TU/e to maintain a list with all of these students. In such scenarios where multiple authorities collaborate we cannot assume that all authorities know all users in the system. For instance a student Alice at RU may not be known to TU/e while Alice wants to use the resources offered by TU/e.

Certificates and Trust Trust plays a key role in collaborative distributed AC. If I trust another authority I can rely on the statements this authority makes, for example statements on the identity or roles of users. To validate a document off-line one would put a signature on the document. Digital signatures aim to bring this notion into the online world. In Chapter 9 we will look in more detail at how to create and check digital signatures. For the discussion now it

is sufficient to know that digital signatures allow each signer's signature to be checked by other parties using the *public key* of the signer. (Consider the public key as the online version of 'your handwriting'.)

Definition 7.1.4.1 *A certificate is a digitally signed statement that the signing party at the time of signing considers to be true (claims/confirms/takes responsibility for/...).*

If Alice trusts Bob (and has his public key) than Alice can also trust statements that Bob signs. The level of trust in the signed statement by Bob, i.e. his certificate, can depend on how sure Alice is that she has Bob's key, how strong the signature scheme is, the level of trust in Bob and the statement that Bob is making. For example, Bob may be an expert on baking so we will trust a bread recipe but not a medical prescription. One may even consider the content rather than the type of statement. For example, trust if the recipe looks 'normal' and do not trust if the recipe has suspicious ingredients (e.g. asbestos).[3]

When Alice is not sure that the public key p is really the public key of Bob, she cannot trust signed statements that are checked using p. When Alice and Bob meet in person they can share their public keys. If Alice and Bob can only communicate over an insecure channel (e.g. the Internet) and Bob send his public key then an attacker Mallory may change it along the way. Thus Alice needs some way of checking that the key is correct. If Alice trusts Charlie who already knows the key of Bob then Charlie could issue a certificate stating 'the public key of Bob is 1234'. As the statement is signed by Charlie, Mallory can no longer change the key without Alice noticing.

In the scenario above Alice grants Charlie the role of a **Certificate Authority (CA)**, i.e. she trusts that Charlie is an authority on the key of Bob. How does she know Charlie is an authority? There could be another authority Daisy that says so. How does she know Daisy is an authority on authorities? This chain can be continued for a while but will need to end as some point at a *root CA* Rob that Alice already trusts without the need for other authorities vouching for Rob.

In HTTPS this approach is used to check certificates which authenticate the website you are connecting to. For example, an (intermediate) *Certificate Authority* TERENA CA checks the identity of TU/e and its public key and signs the statement: The public key for `www.tue.nl` (256 bytes) is

[3]The term trust is used in many settings and in many different meanings. You will also know the term from day to day life. Here we will focus on more technical interpretations of the term trust [6, 13, 21] such as; a trusted statement is one which is supported by certificates.

BB 47 DE 23 5D 66 A1 72 CB C0 36 43 94 75 06 36
39 1B 82 9D 37 B8 CE 9C 3B 68 B7 FC 6A AF F1 03
D6 6E F5 A3 EF 00 1C 9B 9F C1 4D 90 A8 F2 B8 43
F5 9A 2F 83 84 B6 74 8E 81 C8 32 79 47 DF 0D D6
78 91 A4 36 84 10 F7 AC 4B D7 E7 EF 1D A3 BF CA
0A AD C9 9A E5 63 AD 01 0D 32 6D 92 35 81 1B 42
DF 75 F1 88 F6 53 D4 D8 35 B4 8B A5 87 14 5B D0
07 F0 6A 49 45 18 A6 B2 65 41 BC 5F FD 8D 60 10
D4 5B 68 04 44 43 C4 68 0F A8 3D EA E5 90 1B C0
7D E5 8E 9D F8 14 63 50 51 C9 C4 01 6D 11 FF CE
2E 4F 76 18 D0 3B 48 44 66 D8 40 A9 FB 11 04 33
4F CC 84 5E E0 01 0B 62 39 7F 43 DC EF 56 BB 50
E1 8A 99 13 52 AF B8 1B AD 7B 1F 7C 46 BD FC 31
43 27 32 D1 26 08 67 5D C9 32 7D 43 F6 DA BD 85
50 9A 6C 82 39 0E C8 92 57 56 39 3F E5 63 0A A8
D2 96 20 4A 0F 33 83 63 38 57 16 C5 47 9F 20 71

.

However, as Alice does not know TERENA CA and its keys, Alice needs to find out whether she can trust this key and trust TERENA CA to issue such statements. For this the TERENA CA has a certificate from another CA, USERTrust (as shown in Figure 7.7), that validates Terena's key and states that TERENA is trusted to issue certificates for websites. This chain of certificates finally ends at a root CA; a well known root of trust that Alice knows and has the public key of (typically built into Alice's browser and/or operating system).

A risk with this approach is that trust is full and fully transitive; the root and intermediate CAs are trusted to only certify fully trustworthy intermediate CAs and correctly verify the identities of all websites they issues certificates for. However, if one step in the chain fails the whole system can break down. Several incidents show that this can indeed happen. Hacking into the systems of the CA is one way to obtain fake certificates.

For example:

- The recent Turktrust incident.
  - `technet.microsoft.com/en-us/security/advisory/2798897`,
  - `www.theregister.co.uk/2013/01/04/turkish_fake_google_site_certificate/`)
- The Comodo incident.
  - `www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html`
- The well known Diginotar incident (also 2011).

Use of outdated cryptography (such as the MD5 hash) also creates a risk; it has been demonstrated that a fake CA certificate can be created by using MD5 hash collisions (`www.win.tue.nl/hashclash/rogue-ca/`, video at `dewy.fem.`

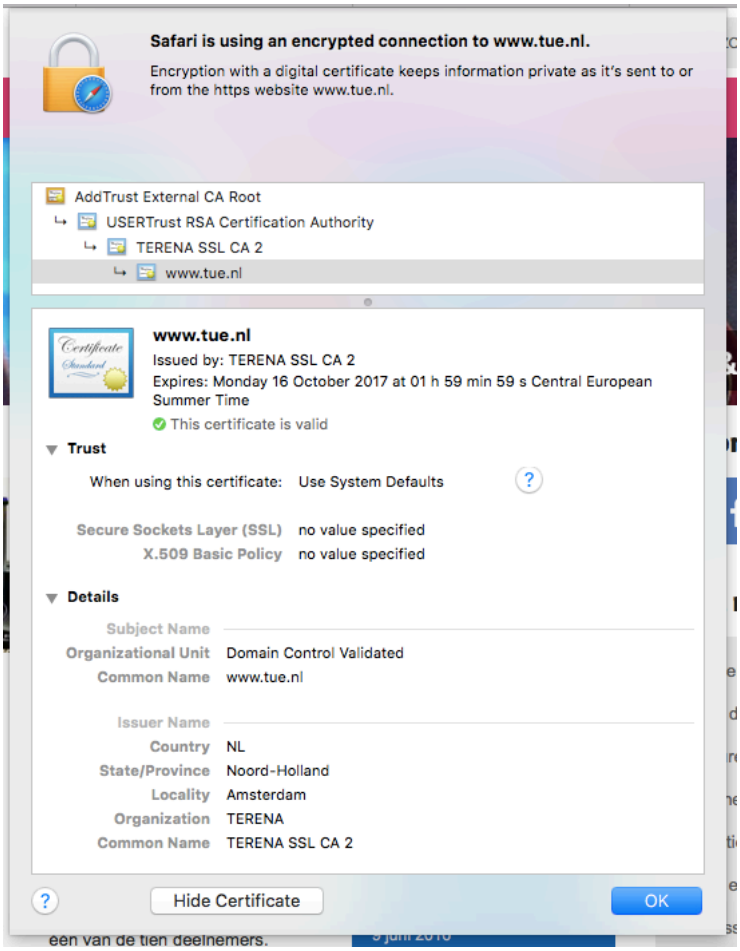


Figure 7.7: The certificate for `www.tue.nl` that contains the public key as viewed by the Safari web browser (the key itself not shown in the image).

`tu-ilmenau.de/CCC/25C3/video_h264_720x576/25c3-3023-en-making_the_theoretical_possible.mp4`).

In HTTPS there are basically only two roles; public key holder (the website we want to visit) and CA that certifies public keys (for anybody, anywhere). If we want to do distributed access control we may want more roles. This would also limit the risk with respect to the problem above; we could have different roles such as a CA for `.nl`, for `tue.nl`, etc. Then we could trust someone for statements about e.g. `www.tue.nl` without having to trust them for statements about `www.bank.com`. One way of doing this is role-based trust management, which extends the ideas of RBAC to a distributed setting.

Role-based Trust management (RT) This management scheme [21], extends the idea of role hierarchies across multiple entities. In RT the policy of an authority consists of a collection of rules of four different possible types as shown in Figure 7.8.

$A.r_1 \leftarrow B$	(simple member)
$A.r_1 \leftarrow B.r_2$	(simple inclusion)
$A.r_1 \leftarrow A.r_1.r_2$	(linking inclusion)
$A.r_1 \leftarrow B.r_1 \wedge C.r_2$	(intersection inclusion)

Figure 7.8: Policy rules defining roles in role-based trust management.

- In RBAC we have a table which assigns roles to users. Here this is done with a list of (simple member) rules. For example, the rule *TU/e.student* ← *Alice* gives Alice the 'student at TU/e' role.
- The notion of role hierarchy from hierarchical RBAC is extended in RT by the (simple inclusion) rule. For example, the relation 'the role Prof is a sub-role of the role Scientific' of Figure 7.3 can be captured by rule *TU/e.Scientific* ← *TU/e.Prof*. The (simple inclusion) rule, however, is much more general. Most importantly it allows specifying relations between roles of different authorities. For example, *TU/e.student* ← *RU.masterstudent* states that anyone with the role masterstudent at RU will have the role (and thus rights) of a student at TU/e.
- Often it is necessary to combine the roles from a group of different authorities. Rather than having to maintain a list of simple inclusion rules we can use a linking inclusion rule. For example, consider that TU/e collaborates with a group of other institutions, collected in the role *TU/e.CollaboratingUniversity*. With a single linking inclusion rule: *TU/e.student* ← *TU/e.CollaboratingUniversity.student* we assign the TU/e.student role to all students at all these institutions. (Note that, while TU/e should define the CollaboratingUniversity role it can use roles of other the parties as in *TU/e.CollaboratingUniversity* ← *VSNU.members*)
- Sometimes role membership has multiple requirements. This is expressed by the intersection inclusion rules. For example, *TU/e.honorCandidate* ← *TU/e.masterstudent* ∧ *TU/e.HasHighGrades* states that candidates for the honor program are master students with high grades. Thus student with a low grade and Bachelors students, even those with a high grade, will not be eligible.

The RT language allows us to make precise statements about our trust in other entities; e.g. we trust VNSU to define universities but not to define TU/e staff members. If the TU/e collaborates at masters level with RU then there is no need to let RU bachelor students have access to TU/e resources. If RU provides a role masterstudent we can include them in our role students; *TU/e.student* ← *RU.masterstudent* captures that the TU/e trusts RU to correctly define master students and trusts master students at the RU with TU/e student resources.

To determine membership of roles in RT we could gather all policies together and compute the roles. However, not all parties may be willing to give their complete policies away. Distributed computation mechanisms have been defined

to compute role membership without parties having to reveal their (complete) policies.

Irrespective of whether we compute results centrally or in a distributed fashion, we have to solve the problem of 'securely'[4] moving statements about roles and role memberships around. Certificates are used for this. The RU can sign the rule 'RU.student ← Alice' and send it to TU/e which checks it using the RU's public key. Then the TU/e can combine the RU rule with its rule 'TU/e.student ← RU.student' to conclude that Alice is a student and give her the corresponding permissions.

Note that we have moved somewhat from discussing authorization and access control to establishing and sharing attributes of the users; i.e. to authentication. In the next section we focus on authentication; establishing attributes of users and sharing these attributes.

7.2 Authentication

Who are you? Depending on the situation you could answer this question by giving your name ("I'm Bob"), giving the group you belong to ("I'm a student"), or by giving the role you are playing (e.g. "I'm the bartender"), etc. Similarly your *identity* in a computer system can be a unique property of you (your user name, bank account number, public key) and/or something you share (being a student at the TU/e) and/or something that you are only part of the time (being a bartender in the weekend). If we take the same perspective as in the attribute based access control above we can answer the question 'who are you' with a list of attributes.

Definition 7.2.0.1 *An identity is a set of attributes that uniquely describe a person within a given context.*

Definition 7.2.0.1, following the NIST (The National Institute of Standards and Technology (NIST), a US federal institute) electronic authentication guide [7], requires there to be enough information in the attributes to uniquely identify a person. For example, if there are two bartenders one wearing a red and one a blue shirt then 'the bartender' is not technically an identity; one would need to say 'the bartender' 'in the blue shirt' (a set of two attributes). However, in the text below we will be a bit loose in the use of the term identity. For example we may use the term identity for a given set of attributes without verifying that these attributes are sufficient to *uniquely* identify an individual.

When you call a close friend you can typically suffice with an "it's me" as authentication; your friend will recognize you from you voice. Someone else, on the other hand, may only be able to recognize your voice after you've reminded them by stating your name. (You, the *claimant* state an identity claim.)

We thus distinguish between *identification*, finding the identity of an individual (e.g. entering a user name), and *authentication*, proving that an individual indeed belongs to a (claimed) identity (e.g. entering a password). The processes of identification and authentication are not always as clearly separated as in the user name - password example; for example when you meet someone you know you do identification and authentication in one step by recognizing their face.

[4] what does this mean here?

Thus you can claim that you have a certain identity (a set of attributes) but can you prove it (how can you authenticate)? How can someone else, a *verifier*, verify that you as claimant indeed have the identity (attributes) that you claim to have. There are three types of *factors* that can be used for authentication:

- what you **have**; e.g. the key to your house, a bankcard, an OV chip card, etc.
- what you **know**; e.g. the hiding place of the spare key, a security code (pincode), a password, your mother's maiden name, etc.
- what you **are**; e.g. your face which is recognized by your roommate, your voice on the phone, your behaviour (e.g. you're serving drinks), etc.

One can of course also combine different mechanisms, e.g. a bankcard with a security code combines the 'what you have' factor with the 'what you know' factor. Such *multi-factor authentication* is often used when there are high security requirements such as for use of your bankcard which represents a large monetary value; the *level of assurance*[5] is matched with the requirements of the application. By using different factors together attacks which target a single factor no longer work; e.g. pickpocketing to steal the 'what you have' (bankcard, a key) is not sufficient, nor is looking over your shoulder to learn what you know (security code, password). Both attacks have to be combined making it much harder to perform.

We thus have the identity (such as 'Bob', 'age 21', 'student at TU/e') of a claimant and ways to check this identity by a verifier; the party that checks whether you have a certain identity, or at least have certain attributes as part of your identity. This is similar to the bouncer at the entry of the club checking attribute 'age 21'. Finally, we have a *relying party* that offers you a service based on your attributes (such as the bartender serving you beer.

Thus far we have discussed authentication in general, but within computer networks we are mostly concerned with remote electronic authentication (**e-authentication**) where end-users prove their attributes to an online service using an authentication protocol. (Recall the importance of establishing attributes for access control discussed in the previous section.)

The first phase in an e-authentication system is a *registration* process where applicants gets issued *credentials* that link their identity (attributes) to *tokens* (things the applicant controls). For example, Alice goes to the student registration desk and shows her passport. Then they check that Alice has paid her tuition fees and issue Alice a student card with her picture on it. Here the student card is a token which is linked (a credential) to the attribute 'student at the TU/e' physically by the design with TU/e logo and electronically by the same statement digitally signed by the TU/e (a certificate[6]). In this example the TU/e asserts the student attribute. Different terms are used for the source of attributes in different settings for example Credential Service Provider, Attribute Provider and Certificate authority. Different naming typically indicates some variations in what they provide and how they provide it. We will not go into that level of detail here. Of course how strict this registration procedure is and how much you trust the attribute provider influence how sure you can be about the assigned identity (level of assurance).

[5] also known as level of authentication

[6] Electronic credentials are usually issued in the form of certificates.

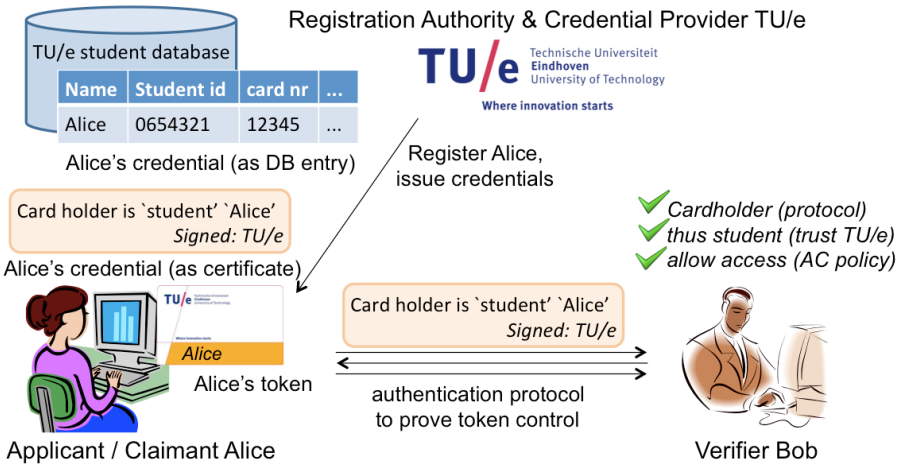


Figure 7.9: Authentication used for an access control decision.

In the next phase a claimant tries to authenticate to a verifier. Physically the student card provides a two factor authentication; someone can check it is your card and thus that you are a student by you presenting the card (what you have) and by comparing you to your picture on the card (what you are). These factors do not translate directly to this phase of the e-authentication; for example a remote verifier cannot see you to compare you to a picture. Instead you authenticate by showing control over tokens and the corresponding credentials then establish your identity (attributes). How control of the token is proven influences how sure you can be (level of assurance) about a claimed identity. For example in electronic banking you prove control over your back card by answering a challenge from the bank that requires you to have your card and know your pin. Therefore, in essence you use two factor authentication to authenticate physically to your token (you have the card and know your pin) which then confirms your identity to the remote verifier. An authentication protocol is needed for the token to confirm your identity to a remote party. Challenge-response and authentication protocols in general are treated in more detail in Chapter 10. (Note that tokens do not have to be physical objects. A password by itself, for example, is considered a token.)

The registration procedure, what type of tokens are used, how they are managed and how control of tokens is proven together determine the level of assurance. The NIST guidelines [7] introduce four levels;

1. For Level 1, the lowest level, there is no need to check the identity during registration as such authentication at this level gives no guarantees about the identity itself, only some assurance that the same claimant is involved. While sending secrets or passwords on the network without any protection (i.e. 'in plaintext') is not permitted, strong cryptographic mechanisms are not required at this level.

2. Level 2 introduces requirements for the registration; the applicant should show, which may be done remotely, possession of a valid ID number (e.g. from a passport, bank account, etc.). The information linked to

this ID number is compared to the current ID application. For example the name in your passport is compared to the name you register under as a student. A single factor token for remote authentication, such as a password (what you know) or a card that does not require a pin to use (what you have), is sufficient.

3. Level 3, multi-factor remote network authentication, requires that at least two authentication factors are used. This could be with a single token using multiple factors such as a card that requires a pin to use or the combination of multiple tokens such as a password and a card without pin. The information to be provided during registration remains the same but now the registration authority must verify the provided information (for example check the bank records against the provided account number).

4. Level 4, the highest level, requires that registration is in person and using a primary, government issued ID (e.g. a passport) with a picture and a second ID (such as a verified bank account number). The registration authority must verify the primary ID at the issuer or at other reliable sources. Non-repudiation of the application must also be provided (e.g. by taking a photo or fingerprint of the applicant); the applicant should not be able to deny making the application. In the authentication phase strong cryptographic proofs using multiple factors are needed to demonstrate control over the token(s).

In practice, the achieved level of assurance is the highest level for which all of the requirements are met. Thus if I use a cryptographic protocol to prove possession of a fingerprint and pin code protected smart card but the card was issued to me without checking who I am the assurance level is at Level 1. If I go through a rigorous vetting procedure during registration but I use telnet to log on (in which I 'prove' my identity by sending an unprotected password over the network) not even assurance Level 1 is achieved.

Passwords - What you know Passwords are a very popular way of verifying a claimant's identity; they are familiar to the user and easy to implement on a system, e.g. requiring no new hardware or complicated programming. They are a key example of 'what you know' authentication. Important questions are

- Do you actually know (remember) your password?
- Are you the only one who knows it?
  - How strong is password authentication?
  - Can no one guess your password?

A problem is that a hard to guess password will likely be hard to remember. The password also needs to be stored on the (remote) system and entered in the system when used to authenticate. Each opens up venues for attacks.

The *entropy* of a secret captures how hard it is to guess. (Equivalently we can describe entropy as 'how much hidden information does it contain' or 'how much room would it take to store/send it in an optimal encoding.'). If the entropy

(expressed in number of bits needed to represent the hidden information) is n then one needs on average half[7] of 2^n guesses to find the secret.

Consider a copier placed in a hallway that uses a 5 digit code to do both identification and authentication. There are 10^5 possible codes thus the entropy may seem to be log 2 of 100.000 and the average number of attempts needed to find the code is thus $1/2 * 100,000$. This could be considered to be enough as very few attackers would try 50.000 codes (i.e. if they get the chance) to be able to make free copies.

In reality the security was a lot less. Random guessing attacks may already be a problem for limited length passwords but attackers can do much better than random guessing. Users typically do not choose their passwords by randomly generating a sequence; this may be a good way to get a password with high entropy, it will likely also result in a password that the user cannot easily remember. Instead users pick easy to remember patterns; such as names, words or keyboard patterns. A smart attacker will thus try those first, significantly reducing the required effort. For the copier example above the first code tried, e.g. 12345, may work. (Note that penetration testing without permission from the resource owner can be considered hacking ('computer vrede breuk') which is illegal in the Netherlands (article 138ab[8] of the Dutch criminal code.)

There are different techniques to combat weak passwords and each has their own advantages and disadvantages. Randomly generated passwords are stronger but harder to remember. Guidelines on how to construct 'good' passwords should not be too specific as that also gives information to an attacker, i.e. decreases the entropy. Reusing the same password on multiple (remote) systems only gives one that needs to be remembered (so perhaps it can be stronger) but means that a break-in in one system (or a malicious administrator[9]) will affect the security of other systems as well. A password safe can be used to store different strong (e.g. randomly generated) passwords on different systems but does create a single point of failure, both with respect to confidentiality (all passwords protected by a single master password) and availability (if safe unavailable or master password forgotten then all passwords are lost).

Passwords have to be checked by the verifier (remote system). But a whole list of unprotected passwords stored on the system makes a tempting target for an attacker (may be useful to break into other systems as well if users reuse passwords.) One could use an access control system or encrypt the passwords to protect the password list but the system still needs access (e.g. the decryption key) to be able to check any password that is entered. The system only needs a way to validate that a claimant knows the password, and this can be achieved without storing a password list: a commonly used method to protect passwords is to only store the *hashes* of the passwords, not the passwords themselves. Hash functions produce as fixed size output for any length input and are one way (irreversible) and collision resistant (hard to find inputs that give the same output). Thus the passwords are now safe(r) as the hash is irreversible. When a claimant provides a password the verifier hashes it and compares it with the hash value stored for the claimed identity. As the hash is collision resistant this is just as good as comparing the entry directly with the identity's password.

[7] Because the attacker may be extremely lucky and find the secret on the first try or be extremely unlucky and find it after 2^n tries.

[8] `http://wetten.overheid.nl/BWBR0001854/TweedeBoek/TitelV/Artikel138ab`

[9] Recall the importance of using the attacker model for security

Finally the claimant has to tell the verifier the password. To prevent an attacker from just trying a lot of passwords the verifier could apply a *rate limiting* mechanism. After a number of incorrect entries the system could block (or just slow down) login for that user (for some time, or until another action is taken such as entry of a much longer PUK code for a blocked mobile phone after three consecutive false PIN entries.)

Overall passwords form an easy to use but not very strong form of authentication. Protecting high value resources with only a password is not advisable. (See also the requirements for NIST level of assertion.)

7.3 Summary

In this chapter we have addressed (e-)authentication and authorization which are key security issues for networked applications. If we try to express these notions in a single sentence we could say that authentication is assigning/establishing, presenting and validating attributes of users while authorization is determining and enforcing the rights of users based on their attributes. There is, however, no clear cut line between the two as they have some inherent overlap; both authentication and authorization deal with trust and management of attributes. Authentication starts at the user; getting an identity or attributes for a user while authorization ends at the resource; linking usage to the attributes. Which of the steps in between, bringing user attributes and resources together, are part of authentication and which of authorization depends on the system and your point of view. Additionally in literature there is sometimes even more overlap; authentication systems may talk about rights associated with the established attributes while authorization systems may talk about obtaining the user's attributes. Wherever you draw the line, with both authorization (access control) and authentication in place we have a means to link users to their rights over a computer network.

If you wish to learn more about Access Control there is the Master course "Principles of data protection" (2IS27). Also the Bachelors course "Legal and Technical Aspects of Security" (2IC70) treats access control both from a legal and a technical perspective.

7.3.1 Literature

Suggested reading

- A Brief Introduction to XACML [10],
- Executive summary (pages v-viii) of NIST-800-63-2 [7],
- Sections 1 to 3 of: An introduction to role based trust management framework RT [13, Sec 1-3].
- Chapter 4 on Access Control of Security Engineering [3, Ch 4],
- The full text of NIST-800-63-2 [7],
- XACML v3.0 core specification [23].

7.4 Homework Exercises

1. Give the AC matrix for the following hierarchical RBAC policy. The hierarchy given in Figure 7.3 is used with the policy in Figure 7.2 extended with: Staff may reserve the meeting rooms M1, M2, Scientific (staff) may book the lab L1. Sandro is a Prof, Heleen is a Financial (staff).

2. Consider the RT policy:

$$
\begin{aligned}
\text{Alice.Friends} &\leftarrow \text{Bob.Friends}\\
\text{Alice.Friends} &\leftarrow \text{Charlie}\\
\\
\text{Bob.Friends} &\leftarrow \text{Alice.Friends}\\
\text{Bob.Friends} &\leftarrow \text{Charlie}\\
\text{Bob.Friends} &\leftarrow \text{Dave}\\
\\
\text{Charlie.Friends} &\leftarrow \text{Charlie.Friends.Friends}\\
\text{Charlie.Friends} &\leftarrow \text{Alice}\\
\\
\text{Dave.Friends} &\leftarrow \text{Eva}
\end{aligned}
$$

   (a) Who are the friends of Alice?

   (b) Is there any friend of Alice that is not a friend of Bob or vice versa?

   (c) Is Eva a friend of Charlie?

   (d) Write a rule that states that friends that Alice and Dave have in common are also friends of Eva. Who fits this description?

   (e) Is it possible to write a rule that states that Alice is not a friend of Dave?

3. What information is missing if we want to use RT policies for authorization and how can we address this?

4. Describe the following scenario and situations (as far as possible) using an access control matrix, a role based access control system, RT and XACML. A Hospital has a patient electronic health record (EHR) system. An EHR describes the medication history of a patient. There are two possible actions on the EHR; read the content and add a new prescription.

   - The hospital has doctors Daisy and Edward, nurses Nancy and Mark and Patients Alice, Bob and Charlie.
   - Doctors are allowed to read the health records of patients.
   - The doctor treating a patient is allowed to add new prescriptions and may let a nurse read the health record of the patient.
   - Daisy is treating Alice and Bob, Edward is treating Charlie.
   - Nurse Nancy is assisting Daisy with the treatment of Alice.

   Give a scenario in which Nancy reads the record of Alice; include the steps involved and what happens in/with the AC system.

5. With PIN payments a two factor authentication of the paying party is performed.

   (a) Which are the two types of factors used and how are they implemented in this example?

   (b) Why are two factors used rather than only one?

   (c) What is the third type of factor and how could it be applied in this setting?

   (d) After entering the PIN code the terminal shows the amount to be transferred and asks for confirmation from the user. What type of attack is this meant to prevent and how effective is this method?

6. Translate the following XACML policy to a textual description of the policy:

```
<Policy PolicyId="SamplePolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
<Target>
 <Subjects> <AnySubject/> </Subjects>
 <Resources> <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
  <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
 </ResourceMatch> </Resources>
 <Actions> <AnyAction/> </Actions>
</Target>

<Rule RuleId="LoginRule" Effect="Permit">
 <Target>
  <Subjects> <AnySubject/> </Subjects>
  <Resources> <AnyResource/> </Resources>
  <Actions> <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
   <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                          AttributeId="ServerAction"/>
  </ActionMatch> </Actions>
 </Target>

 <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
   </Apply>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
   </Apply>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
  </Apply>
 </Condition>
</Rule>

<Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

7. Rate limiting to counter password guessing, like most security mechanisms, introduces a trade-off. Which security property is diminished and which is increased by the introduction of rate limiting?

Chapter 8

Network and Web Security

On a computer network with different interconnected systems security is not only important, but also hard to achieve. Not only do we need to consider threats on our own (local) system but also on all systems connected to it, as well as the connections themselves. Where we may have some trust in our own system we likely will not trust all systems on the network and their users. The interests of the other parties on the network may be completely different than ours. As we have seen in our security analysis, conflicting interests lead to (security) risks. In this chapter we will look at some of specific computer network (protocol) related threats and corresponding countermeasures.

In our discussion we try to answer the following questions:

- Why are networks and web applications so vulnerable?
- How to achieve network and web security goals?
- How to approach different attacker models?

We start by looking at threats at the lower network layers, and move up the protocol stack illustrating that risks exist at each layer and also in moving between the layers. Next we touch on (distributed) denial of service attacks that are focused on breaking the security attribute 'availability'. We then move to the top of the network protocol stack and we treat the application layer separately, looking in particular at web services and related vulnerabilities.

Having described the threats we discuss some key network security technologies aiming to address these threats such as security protocols and intrusion detection and prevention systems (firewalls, intrusion detection systems, virus detection, etc.).

8.1 Network Layers and Corresponding Threats

In this section we look at threats at and between different network protocol layers, working upward from the lower layers towards the application layer. Threats at the application layer itself is the focus of the next section.

Of key importance when considering the threats and attacks described in this section is to keep in mind who and where the attacker is and what she is

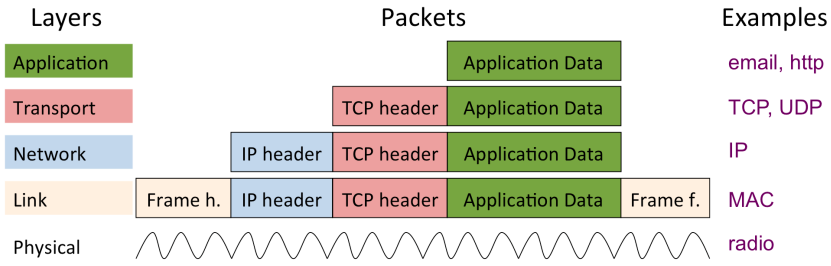


Figure 8.1: Simplified OSI Model for TCP/IP layers

trying to achieve; i.e. what is the **attacker model** that belongs to the threat. Different attackers will have **different capabilities**. For example, an attacker connected to the same hub will see all the messages being sent. An attacker on the same, trusted, local area network (LAN) will be able to perform attacks without having to worry about an Internet firewall protecting the LAN. Attacks by different attackers will also have **different goals**. An attack may be aiming to gain a capability, for example to get messages onto the LAN. Obviously such an attack is only relevant for an attacker that does not yet have the capability; only an outside attacker would need an attack for getting messages onto the LAN as an inside attacker already has this capability.

To secure the network we need to consider attacks at different layers (see for example Figure 8.1). Consider the network layer model for TCP/IP (see Figure 8). An application process will use the transport layer's connection service to manage a connection with a remote process it wants to communicate with. But to do this (human understandable) addresses used by the application, such as `www.tue.nl`, need to be translated to IP addresses understood by the network layer using DNS. An attacker may try to get the traffic redirected to their IP address by disturbing this step (e.g. through DNS spoofing). Alternatively, the attacker could influence lower layers to achieve the same result. For example, an attacker could eavesdrop messages if she has access to the physical layer.

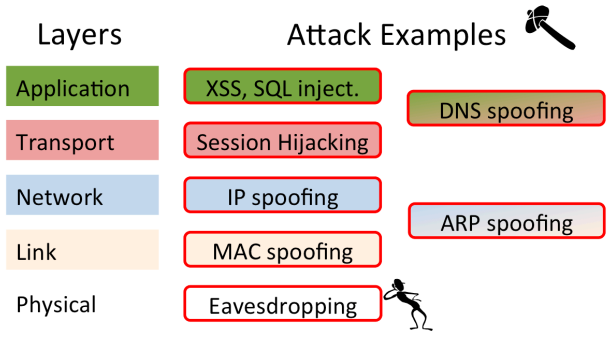


Figure 8.2: Examples of attacks at and between different layers

MAC As we discussed in Chapter 6, a media access control (MAC) protocol is a link layer protocol and a MAC address is used to identify each network device

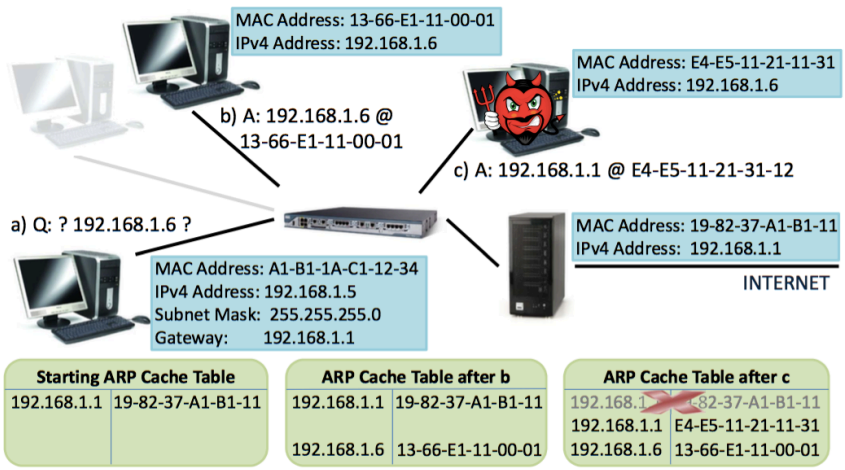


Figure 8.3: ARP spoofing in action

interface. Some wireless routers (access points) use MAC address 'whitelisting' as a security mechanism, allowing only access by the listed MAC addresses. However, a network device can claim to have any MAC address as modern operating systems allow MAC address changes in software. Some wireless routers even support setting the MAC address that they use as some Internet modems only talk to one fixed MAC address. Installing a new router would thus not be possible without spoofing the MAC.[1]

ARP: IP to MAC On the network layer IP addresses are used to identify devices on the network. Routing of IP packets depends on whether the IP is local or not (as configured by the subnet mask). Datagrams destined to local subnet addresses are directly sent to the MAC address of the machine that belongs to that IP. Other datagrams are sent to the gateway responsible for passing messages to the rest of the network. If machine A does not have the MAC address that belongs to some local IP-B in its cache it will use the address resolution protocol (ARP) to broadcast the question 'what is the MAC address corresponding to IP-B?'. The machine with IP-B will respond by sending 'IP-B is at MAC address x'. For efficiency machine A typically does not remember whether or not it asked for IP-B. Instead, whenever it sees the message 'IP-B is at MAC address x' it updates its cache. Thus to claim an IP-B all that is needed is to send a fake response (even without a request).

What can be done to prevent ARP spoofing? One thing to note is that there are some legitimate uses where the redirection of traffic is actually intentional, not an attack. For example, a backup server may transparently take over the

[1]In a way this means that the routers are designed to circumvent a security feature of the modem; some ISPs disallowed the use of multiple devices on the same connection in the early days of broadband internet though mainly to prevent commercial use or sharing the connection with other households. Consumers having multiple PCs, let alone other devices with internet connection, was not very common back then. Technically this could thus be classified hacking ('computervredebreuk' in Dutch) - for this type of considerations see the electives course Legal and Technical Aspects of Security (2IC70).

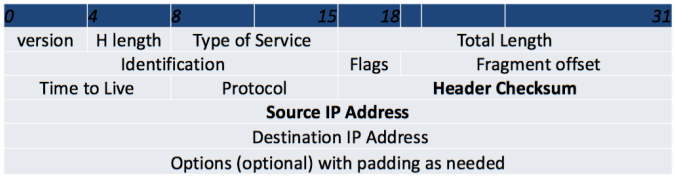


Figure 8.4: IPv4 packet header

role of crashed server by claiming its IP. Other uses may be to redirect a new machine on the network to a sign-up page before giving it access e.g. to (the gateway and) the Internet.

Tools which monitor the network can be used to look for fake responses (e.g. responses without requests, multiple responses to a request), poisoned ARP caches (e.g. different values for the same IP), etc. To limit possible damage one could also use static entries for key addresses (such as the gateway, important local services) and not use ARP for these addresses. A disadvantage here is the maintenance involved; if any of the static addresses change, all devices on the network have to be updated. Also, recall that any device can claim to have any MAC address so using the correct MAC address does not guarantee that the message goes to the correct machine.

Instead of trying to prevent the ARP spoofing we can try to solve this at the higher network layers, taking into account the fact that the lower layer may be unreliable in the design of protocols for the higher layers.

IP Like the MAC address, the IP address is just a plain text string inside the message. Spoofing an IP address in a message is simple; just change the IP header of the message. (See Figure 8.1: put the IP address you want and compute the corresponding header checksum.) The message will then appear to come from (or be intended for if you change the destination) that IP address.

To help mitigate attacks based on IP spoofing one can use firewalls to block message from outside the network that claim to come from an IP on the local network. Also, it may be possible to trace back the source of the message (the routers forwarding the message may allow) which may help finding the source of the attack, or at least defend against it closer to its source (which may be important e.g. in denial of service attacks). Note that the attacker model has changes compared to the discussion on e.g. MAC. With MAC spoofing we consider an attacker on the local network. IP spoofing is basically possible for any attacker. The firewall defense clearly only makes sense when we are dealing with an outside attacker. (As we have seen above an attacker on the local network can likely mount a more powerful attack, actually claiming the IP rather than only doing IP spoofing.)

IPsec is a set of protocols for authentication and encryption of IP packets. It supports mutual authentication of the agents involved. **Transport mode:** to protect confidentiality, the content of a packet is encrypted. The header is not modified (not to influence the routing practices on the network). The integrity of the packet payload and parts of the packet header can also be protected. **Tunnel mode:** the entire packet, including header, is encrypted and then the result is sent as the payload of a new IP packet.

IPsec was developed as part of the IPv6 standard (though it can also be employed in IPv4), but not all implementations of IPv6 include IPsec. *Security associations* are basically keys used to communicate, along with algorithms, protocols and settings used. IPsec uses Internet Key Exchange (IKE) to set up security associations. It uses the Diffie–Hellman (DH) key exchange to set up a shared session key (or to be more precise a shared secret from which keys are derived). It can also use certificates to authenticate parties.

When you initially setup an IPsec connection, you have some confidence that you will securely communicate with the party that you setup the connection with; the confidentiality and integrity of the communication content can be protected. However, an authentication phase is still needed in which you ensure that the party you setup the connection with is actually a party you want to communicate. This introduces issues of policies, managing secrets (e.g. keys), etc. which we will see returning in following lectures.

TCP The TCP protocol uses sequence numbers to identify blocks of communication and these sequence numbers form a basic form of authentication; packets are only accepted by the receiver if the right sequence number is used.

If we can predict (see e.g. [5]) the (initial) sequence number that the server will use, we can use a spoofed IP and guessed sequence number to start our own session with the server; we don't see the responses but we can issue requests which will be accepted as coming from the (trusted) client. We do need to prevent the real client from reacting to messages as seeing an unexpected session number from the server will cause it to send a reset request which would break our connection. Note that this attack is more a fake session initiation than a session hijacking attack but is typically seen as belonging to the category of session hijacking attacks.

Clearly if we can somehow get the communication with the server we can simply read the messages and the sequence numbers within them. (We could for example pretend to be a gateway between the server and the client networks with the techniques described above or we could also try to get responses from the server to be sent back by using source routing; in which the sender of the packet can indicate (part of) the route that the return packet should follow. However, source routing is nowadays usually disabled.)

A session between a client and server may start with authentication of the client. In this case we may want to actively take over an (authenticated) session rather than try to start our own new one. If we can see the traffic to/from the server we can wait for a connection to be created and authentication to complete, eavesdrop the session numbers used and then inject our own messages with IP spoofing using the eavesdropped session numbers. Variants of this attack differ in how they deal with the client and server responses. For example we can take down the client after the session has started to prevent it from interfering with our stolen session. We could also desynchronize the client and server by resetting the connection on one end but not the other; we now act as a man in the middle forwarding the responses of the client and the server (adapting the session numbers) when we want and changing the messages as we please. If the session includes setting up encryption keys then this type of a man in the middle attack can also break protection at higher network levels that rely on this encryption.

Figure 8.5: Basic operation of DNS and DNSSEC.

A problem with some variations of the attack can be so called 'ACK storms' which can give away the attack. The server acknowledges the data that we send to it but will of course send its acknowledgment packets to the client. If we don't prevent these from reaching the client it will, upon receiving these packets, notice that the acknowledgment number is incorrect and send an ACK packet back with the correct sequence number in an attempt to re-establish a synchronized state. However, this response itself will be seen as invalid by the server, leading to another ACK packet, creating a loop that continues until one of the ACK packets is lost in transit.

DNS: Domain (URL) to IP DNS (amongst other its other services) translates human readable addresses (domains), such as `www.tue.nl` to IP addresses. A client (e.g. a web browser) typically keeps a local cache (in the brower itself or in the OS running the browser) of the Domain-IP mapping. However, for new domains it will need to contact a name server. A local name server is typically set along with the IP address of the machine (manually or as an optional part of DHCP). To look up the address, this local name server will recursively make other calls if they are needed to find the address as we discussed earlier in Chapter 3. The local name server has a cache of its own.

The example in Figure 8.5 shows a possible flow of events. If (1) the client tries to look up `www.tue.nl` and (2) this domain is not in the cache the name server will (3) ask the root name server. A 16-bit ID is included in the query and an answer is linked to the query by checking its ID which should be the same. The root name server does not have (4) the IP for domain `www.tue.nl` itself but does know which name server is responsible for the top level domain `.nl`; lets say this is for example `ns.sidn.org` for the `.nl` domain. Thus the root name server refers the requester to this server (5). This reference is typically given by 'name'. Here the name server already knows (6) where to find `ns.sidn.org` from its cache (otherwise it would first have to do another DNS lookup). Thus it contacts (7), the server which again will do a redirection (8), now to `ns.tue.nl`. Here the name server for the domain is actually within the `tue.nl` domain. While `ns.tue.nl` is responsible for (known as the 'authoritive name server') the domain `tue.nl`, including `ns.tue.nl`, if we do not know the IP for `ns.tue.nl` we can of course not ask it for its own IP. To solve this the .nl name server stores the IP address of `ns.tue.nl` (a so called glue record) and also sends this with its response in (9). This response is stored in the cache (10). Finally we get the IP we are looking for (11,12) which is stored in the cache and returned to the client (not depicted).

There are several types of attacks possible on the DNS scheme. Figure 8.6

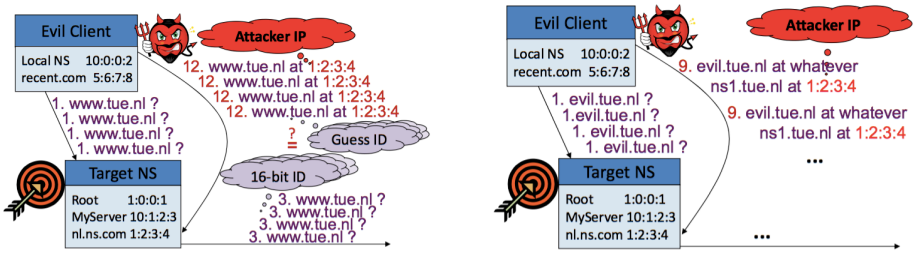


Figure 8.6: DNS poisoning attacks against a local name server.

illustrates, using the same setting as the example in Figure 8.5, two attacks in which the attacker is a client who is trying to 'poison' (i.e. insert a fake record into) the cache of a name server (Target NS). In the first attack the attacker sends a lot of requests for the address it wants to poison. If the address is not yet in the cache, the name server will send out requests for this address. The attacker in the meantime sends many fake replies to this request containing a fake (attacker controlled) IP and using different values for the ID. If one of the IDs used by the attacker matches one of the IDs used in the request of the name server the response will be accepted and the attacker's IP address will be linked to the website (`www.tue.nl`) in the cache of the name server. Other users of the same name server will now be directed to the attacker when they try to visit (`www.tue.nl`).

In a variation of the attack above, the attacker asks for a non-existent domain. The advantage is that the domain will for sure not be in the cache of the local name server and no name server will resolve the domain so the attacker has no competition for its response queries. But what use is getting the name server to have a false record for a non-existent domain? The clue is the use of glue records; if the attacker can get the name server to accept her response she can include a glue record which will also be stored and can make an important domain name (e.g. that of a name server `ns.tue.nl`) point at her IP.

Not only the name server but also the client itself may be targeted. For example (see Figure 8.7) an evil web site could include many items from the `www.tue.nl` domain. If the client visits the page this will cause many requests for `www.tue.nl` and the attacker can send many responses at the same time, hoping to match the ID of one of the requests.

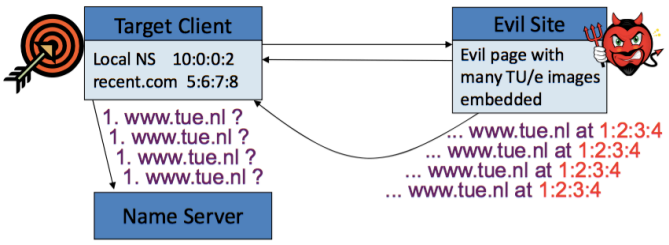


Figure 8.7: DNS poisoning attacks against a client.

DNSSEC One possible defense against the DNS attacks described above is to authenticate responses using digital signatures.
(*Side Note*) Digital signatures are important. The signer encrypts his message or a hash computed from this message with his *private key* (only known to the signer) to create a signature that is communicated together with the original message. The result is a packet that contains the original message plus the signature, called a 'signed' message. The receiver can use the public key of the signer (known to everyone) to decrypt the signature and compare the outcome with the hash computed again at the receiver. This is shown in Figure 8.8. (*End of Side Note*)

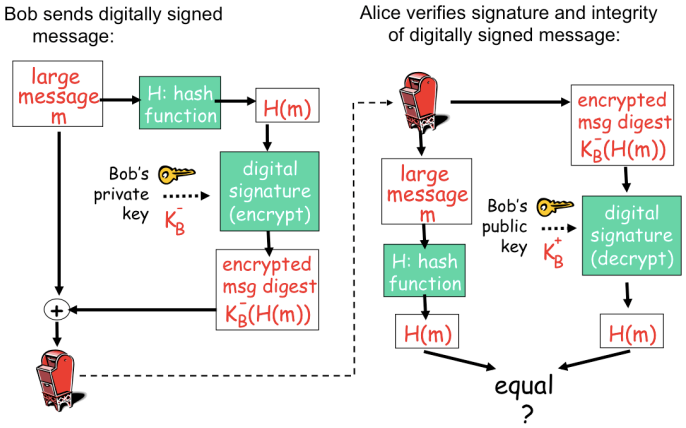


Figure 8.8: Digital signature usage: Note that Bob's private key should not be reproducable from his public key. (Figure by Kurose and Ross)

In DNSSEC (Domain Name System Security Extensions) the name server can digitally sign its response (steps 5, 9 and 12 in the scenario of Figure 8.5). Responses without a correct signature are not accepted so the attacker cannot create a fake response. To check a signature we need the public key of the signing entity. The final reply (12) will be signed by the TU/e name server thus we need the correct public key of this name server. Each name server will know the public key(s) of the root name server but of course they do not have the public keys of all authoritative name servers. Thus we need to retrieve and check these public keys somehow. The way to do this is to have the name server that refers us to the next name server also validate its public key by signing it. Thus the .nl TLD name server, which we trust for telling us the correct name server for the tue.nl domain, will sign the public key for that domain as well. The root name server in turn will sign the key of the .nl TLD name server and this we can check because we have the root public key. Of course if any step in the chain does not deploy DNSSEC the system fails; it thus needs to be widely deployed before bringing real benefits. This and added complexity, key management issues and potential information leakage (e.g. which names exist in a domain) that could be caused by DNSSEC have hindered deployment, e.g. only since 2010 has DNSSEC been deployed at the root level. Though the rate of use is increasing, it is far from ubiquitous; it was claimed[2] that it was

[2] e.g. `http://www.circleid.com/posts/20140908_dnssec_adoption_a_status_report_`

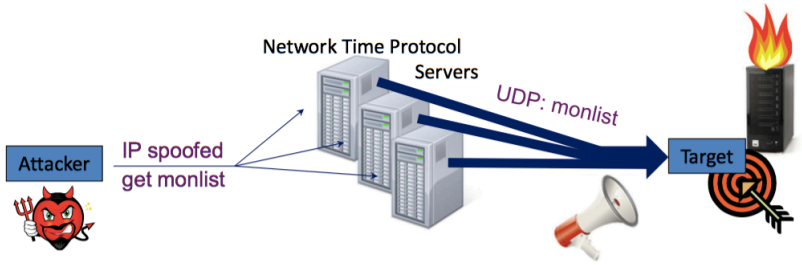


Figure 8.9: Denial of Service attack amplification through NTP servers.

used in approximately 12% of DNS queries as of 2014.

8.1.1 Denial of Service (DoS)

In DoS attacks the attacker aims to break the security goal of 'availability'. There are different strategies that the attacker can use for this. *Logical attacks* may exploit flaws in the networking services of a machine. For example, the 'ping of death' sends a malformed ICMP (ping) message to a target machine causing a buffer overflow that could result in a crash of the target. Of course this is not limited to ICMP. The latest (at the moment of writing) patches of, just to name a few examples, Windows, OpenSSL and GnuPG each addressed flaws which could be exploited in denial of service attacks. (We will see more on logical attacks, though typically against other security attributes such as 'secrecy' and 'integrity', in the next section which treats web service security.)

Another approach the attacker could use is *flooding* the target with more traffic than it is able to handle. This then becomes a game of resources; how fast the attacker is able to send requests versus how fast the target is able to deal with them. If the attacker has much more bandwidth than the target simply sending lots of network traffic may already work. Dealing with a correct looking application layer request may take significant effort by the target making sending many of such requests a potentially more efficient way to occupy the resources of the target machine. The effort for the attacker per request is typically far less allowing the DoS with less use of attacker resources.

Attack amplification typically tries to reduce the attacker effort needed compared to the effect on the target machine. One example of attack amplification is the 'smurf' attack. The attacker sends an IP spoofed ICMP (ping) message to the broadcast address of a (mis-configured) router which will then send this message to all machines in its network. All these machines will now send their response to the IP which was spoofed to be that of the target; a single message by the attacker leads to many being sent to the target. Another example of amplification using size rather than number of messages exploits an administrative feature of Network Time Protocol (NTP) servers. These servers can be asked to return the 'monlist'; a list of the last 600 IP addresses that connected to the server. IP spoofing the request will result in a response, which is much bigger that the request, sent towards the target machine.

`part_one/`

Distributed Denial of Service (DDoS) The resources of an attacker need not be limited to a single machine. A coordinated attack in which many attackers (or many subverted machines such as a botnet under control of the same attacker) collaborate by each sending many requests can overwhelm a target. Simply sending a huge amount of otherwise legitimate requests can exhaust the resources of the target making it unable to respond to requests from others. (Note that the two amplification attacks mentioned above are forms of DDoS.)

While routers of the traffic can be assumed to be willing to help prevent attacks a firewall on a router close to the target will have a hard time to protect the target. Simply blocking all traffic does not help - the target becomes unreachable which is exactly what we are trying to prevent - and distinguishing attack traffic from normal traffic will be difficult. Trying to do so will overwhelm the router creating a possible even bigger problem (everything near the target may also become unreachable).

A solution would be to ask routers closer to the targets to start filtering the traffic. If the attack uses specific type of request one could block that type of request or one could simply block traffic from subnets creating an unusually large number of requests. (Others on the blocked subnet would be affected but at least the target remains reachable from the remainder of the network.)

Yet finding where an attack comes from and thus which routers to ask to do the filtering is not trivial. While traceroute programs allow determining which route packets will take to a certain destination IP address, the source IP in the requests is most likely spoofed and not the IP of the real attacker(s). In stead we need 'IP traceback'; a way to find out where a (stream of) packets is coming from. Simply adding the address of all routers that a packet passes creates too much overhead. Different schemes have been proposed for routers to insert extra information into parts of the IP header that are not needed for a given packet. Given enough traffic the added information enables finding which routers the stream has passed through.

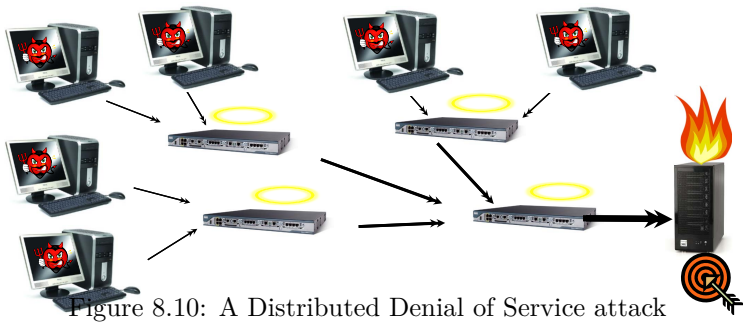


Figure 8.10: A Distributed Denial of Service attack

8.2 Application Layer Security

Above we focused on the mechanisms of the communication at the lower network layers where many of the attacks are related to the packet headers; e.g. IP spoofing by using incorrect information in the IP packet header. In this section we will consider the application layer. At the application layer the payload

becomes more important; we have already seen some examples such as the 'ping of death' where an oversized payload causes an error in the application handling the ping messages. While other layers simply treat the payload as a binary blob to be transported, at the application layer the content gets processed and interpreted. As the content may come from untrusted parties this opens up the risk of attacks through content that is interpreted incorrectly. Thus here we will look at the content of the communication, for example for security of web services we need to carefully consider user input to those services.

Web services need to collect information from the user to do their job; the news article to display, the address to dispatch an order to, etc. Information collected, through setting parameters in the links or explicitly entered in forms, can be manipulated by an attacker. Sometimes changing the user-id in the address bar is already enough to get at the content of another user. Forms used to gather user input may contain checks (e.g. is an email address format correct, does the text entered in the telephone field contain only numbers, etc.). However, on the server end one should not rely on such checks as an attacker can simply change the data after the check or send its own (unchecked) data without using the form at all.

```
<?php mysql_connect('127.0.0.1', 'dbuser', 'mysqlpwd');
  mysql_select_db('name_tabel_mysql');
  if(isset($_POST['login'])) {
    $user = $_POST['name'];
    $pwd = $_POST['password'];
    $query = mysql_query("SELECT uid, uname FROM
      users WHERE uname = '$user' AND pwd = '$pwd' ");
    ... } ?>
```

Figure 8.11: SQL injection vulnerable script and stored XSS.

SQL injection The information used by the website is typically stored in an (SQL) database. To supply the correct information to a user, the web application will construct a query to this database using input from the user. For example, an application (see also the example in Figure 8.11) may look up the record for a given username-password combination resulting in the script line:

SELECT uid FROM users WHERE (uname='$username') AND (pwd='$password')

where users is the name of the table storing user (uname) and password (pwd) combinations indexed by user ids (uid). When Alice enters her username and password the query

SELECT uid FROM usertable WHERE (uname='*Alice*') AND (pwd='*1234*')

is sent to the database and as a result Alice's uid is selected. In the next step documents belonging to that uid are displayed to Alice. If, however, an attacker enters strange and unexpected data, the meaning of the constructed query can change - for example:

SELECT uid FROM usertable WHERE (username='*Whomever' OR '1'='1*')
AND (password='*Whatever' OR '1=1*')

actually retrieves all uids and thus may result in everyone's documents being shown to the attacker. This is a basic example of an SQL injection attack; via the user input additional SQL code is injected into the query.

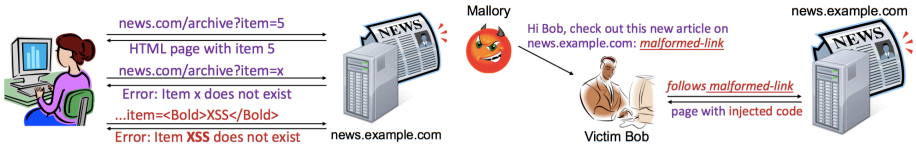


Figure 8.12: Cross Site Scripting (XSS).

When doing a blind injection; where the structure of the database and/or the queries constructed by the web application are not known, some effort is first required to find this structure. Some web applications, when getting an error from the database, simply display the error to the user. This may be useful to the application developer for debugging purposes but will also provide a lot of valuable information to the attacker. For example, the query used, tables that exist/do not exist, etc. Sometimes the application filters the error only showing that some error occurred. Even only being able to see whether an error occurs, or even only how long it takes before the web application responds gives a way to extract information about/from the database.

What can be done to prevent SQL injections? The web application can employ several countermeasures. The main one is input filtering. A basic step could be to check the size of the input; e.g. if the input is a pincode it should only be a few characters long. This would already make it more difficult to construct successful SQL exploits. A stronger defense would be to ensure that the user input is not interpreted as being part of the query by checking the input from the user for special characters (e.g. ') or sequences. The database API may actually have a filtering function that achieves this which should then be used so the web developer does not have to write to code for it (with a risk of errors and/or oversights); for example using mysql_real_escape_string($_-POST['naam']); instead of just $_POST['naam']; in Figure 8.11. Of course this needs to be done for all user inputs in all scripts. This is a main reason why these vulnerabilities still exist; even if the script developer is aware (not all are), it is easy to forget it just once which may be enough to completely expose the database.

When supported by the database, a way to safely get the input to the database is to use parameterized queries. Instead of building a query string which includes the user's input, the query contains parameters and the user input is passed to the database as values to be used for the queries. In this way no confusion is possible between what is part of the query and what is user input. Thus the user input cannot change the query/inject code into it.

In addition to the input one could validate the output of the query. Similar to input size checking we can check whether the output matches our expectation; if we are expecting a single record and get back a whole list something is likely wrong and the results should not be used.

XSS Cross Site Scripting (XSS) is another example where user input forms a danger to web applications and their users. In an XSS attack an attacker gets a user('s browser) to run code that seems to come from another (trusted) website. User (attacker) input to the website is used to to inject code (a script) into a 'trusted' page. Coming from a trusted page, a victim's machine will

allow execution of the code, which can then e.g. control communication with the trusted site (become a 'man in the middle'), steal information only meant for the trusted site (private/confidential information, credentials, sessions keys, etc.) or exploit local vulnerabilities to gain even more access to the victim's machine.

Consider a subscription to a new website 'news.example.com' (see Figure 8.11) where users can place comments with an article which are then shown to all users reading the article. An attacker Mallory could post a comment containing a script which would then be accepted by user Bob's browser as part of the 'news.example.com' website and thus has all the rights any script from news.example.com would have. For example, it can read the cookies that news.example.com has set when Bob logged in and send them to Mallory. With those cookies Mallory can impersonate Bob at news.example.com.

A comment section is an obvious user input and news.example.com may, as with SQL injections employ input filtering, disallowing scripts (see also countermeasures below). However, there are also other, less obvious, user inputs that can be employed to launch an XSS attack as in Figure 8.12. To indicate the selected article, news.example.com uses the 'item' parameter, which should be set to the id number of some item. If a non-existent id number is used the website will report that the given number does not exist.

The error message is a way for a user (e.g. Mallory) to inject code; by setting item to some code instead of an article id that code will be injected into a news.example.com page showing the error message. The script can be used to hide the error message part so the page looks like a normal page. Now Mallory can run code that seems to come from news.example.com but it runs on her own machine under her account. To get the code to run on Bob's machine she needs to get Bob to follow the link she has constructed.

If Bob is well educated in security and is thus suspicious of 'untrusted' links. Mallory will need to hide the fact that the link contains a script. She could display a link different from the actual link and combine this with tricks like: `news.example.com/archive?dummy="This-very-long-argument-....-hides-the-rest-from-view"&item=<script>...</script>` which abuses that fact that Bob's browser will likely only show the first part of a very long link. She could also try to encode it; `%3C%73%63%72%69%70%74%3E` is the same as `<script>` but may not be recognized as such both by Bob as well as by a naive input filter.

Other than educating the user not to follow untrusted links, even if they seem to go to 'trusted' sites, there is little that can be done on the user site. When visiting a page with a stored XSS or after Bob follows the malformed link, Bob's system is confronted with code that comes from the 'right' (trusted) server. Distinguishing it from code that is legitimate is very challenging. (See also Section 8.3 below.)

On the server side, as with SQL, input filtering is an important countermeasure. User input with special meaning should be translated into a 'safe' format. When coding a web application one should thus be very careful with user input. Note that filtering requires being able to distinguish between valid input and harmful content. This may be difficult, for example for a webmail application where the input is an email written in html. Removing dangerous parts but still allowing the user to give all the types of input can thus be very difficult. Web scripting languages typically provide some functions, e.g. the 'htmlspecialchars'

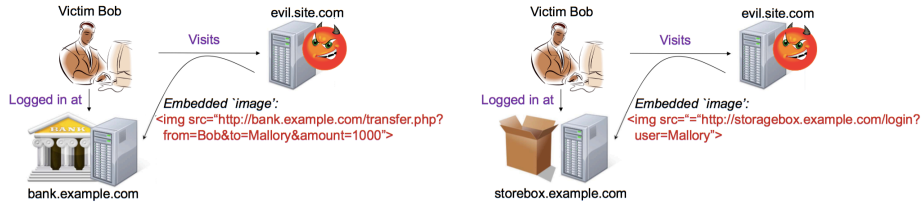


Figure 8.13: Cross site request forgery (XSRF).

function that replaces characters by their html encoding. This, however, is not sufficient to make the user input 'safe' in the example. Additionally XSS input filtering, as with SQL filtering, needs to be done on all relevant user inputs in all scripts.

Tool support to analyse scripts for vulnerabilities such as SQL injection and XSS is thus desirable. There are tools which try to detect XSS vulnerabilities. Such tools aim to ensure functions such as 'htmlspecialchars' are applied in a sufficient way to make the input safe. As automatically checking user input usage against what is dangerous in a given setting is not easy such tools often miss vulnerabilities and/or create many false positives making them less convenient to use. Similarly there are tools for SQL injection analysis as well as more general code analysis tools to search for (security) bugs.

XSRF Cross-Site request forgery (XSRF) is another attack where user input (in this case requests) is misused. While an XSS attack tricks a user('s browser) into trusting and running code, an XSRF attack aims to send authorized but unintended requests to a (trusted) website. Consider a user visiting evil.site.com while still logged in to bank.example.com. (See Figure 8.13.) On the evil site there is a link to an 'image', located at url `http://bank.example.com/transfer.php?from=Bob&to=Mallory&amount=1000`. The user's browser will follow this link to load the image. However, the link is not really an image but a request to bank.example.com to transfer money. As Bob is already logged in at bank.example.com the request will succeed.

A variation of the XSRF attack above, also shown in Figure 8.13, has the same setting of Bob visiting evil.site.com while logged in, but this time at storagebox.example.com where Bob stores his important files. Again a fake link will be followed by Bob's browser resulting at a request to storagebox.example.com to login as Mallory. Bob's session is thus replaced and he is now logged on as Mallory. Of course Mallory does not store her important files here. Instead she hopes that Bob will not notice the change before uploading an important document. The document will go to Mallory's account so she can also access it.

In this section we've seen several types of attacks against web servers and their users. A general defense against these types of attacks is to (try to) decrease the value of what can be stolen; e.g. accepting authentication cookies only from the right IP would make abuse of stolen cookies harder. We will look at network defenses in general in Section 8.3. As a general conclusion for this section we can state: data is more dangerous than you think. Not only unknown programs or devices but also any data from an untrusted source can be a source of attacks.

8.2.1 Malware

Above we have looked at some specific vulnerabilities of networks and the machines on a network. Different types of *malware* exploit such weaknesses to infiltrate the system, replicate, spread and achieve some malicious goal.

Trojans are legitimate looking programs (or other content) that actually carry malicious code inside. *Viruses* can replicate, usually involving some action like running an infected program (like biological viruses need a host organism, computer viruses infect programs). *Worms* are able to replicate by themselves, without the need of human action. Well known worms, such as *conficker*, spread around networks (e.g. the Internet) exploiting vulnerabilities of network services and machines. *Adware* is a type of software that automatically shows advertisements or redirect your searches to advertising websites, in order to generate revenues for its creator. (This is not necessarily malicious, however many types of adware try to hide on your system, avoid uninstall and/or gather information without your consent.) Classification of malware into these categories, however, is sometimes difficult and terms are commonly mixed, using e.g. virus for any type of malware.

While some hacks, viruses and worms may have been 'just to see what I can do', i.e. idealistic, to demonstrate the vulnerability or simple vandalism, modern malware is for the most part targeting big businesses or are even used for digital warfare. The conficker worm, for instance, installs *scareware* (showing pop-ups to get user to buy a fake anti-virus solution) and creates a *botnet*; a network of machines under control of the attacker, which can then be used for sending spam, distributed DoS attacks, renting out to others, etc. The worm has an update mechanism used to download its software for its malicious activities as well as updates to its spreading mechanism and protection against updates of anti-virus software that would be able to protect against it.

(Zero-day) vulnerabilities, exploits, virus building kits and botnets are commodities that are traded on the black market. One thus does not even have to create one's own malware or botnet; it is possible to buy or rent infected machines. (See for example Metasploit `www.metasploit.com` for hacking made easy.) Botnets are controlled through command and control centers. By using a C&C center to drive a botnet, the malware can easily be updated and adapted, making it hard to take down the network of bots. To prevent the C&C center itself from being taken down, it is located in countries where there are no laws to easily do this, its location is hidden e.g. within a list of addresses, using anonymous services in TOR[3] and/or redundancy is used so a new C&C can easily be created at a different location. As in many security areas there is an arms race between taking out botnet C&C centers and new infections, botnets and control methods appearing.

Of course with all the value a botnet represents there are those that will try to take it over, either to dismantle/study it (e.g. [24]) or to use it for their own illegitimate agenda. It has also been suggested to take this defense strategy a step further; use weaknesses in infected machine to force installation of patches, removing of and protecting against malware but there are many moral, legal, practical and technical issues with such an approach. Related to this is use of 'hacking' by authorities which is also the subject of debate what actions are justified, should e.g. 'hacking' by the police be allowed (and if so to what degree,

[3] `www.torproject.org`

in which circumstances)? For example German federal court rejects[4] hacking by police in general though leaves open some possibilities to do so for example with respect to threats that endanger public safety such as terrorism.

APTs and Stuxnet An important trend in the malware landscape is the move from 'blunt tools' doing 'catch what you can' attacks against arbitrary targets to sophisticated, specialized attacks against specific targets; e.g. for data theft, (industrial) espionage. Such Advanced Persistent Threats (APTs) are at the 'frontline' of the (anti-)malware arms race.

Not only the PCs or ICT networks are potential targets for attacks. Attacks on critical infrastructure can be especially harmful. The so called SCADA (supervisory control and data acquisition) systems that manage industrial processes, power plants, locks, bridges, prison cell doors, etc. are vulnerable to attacks.While for example gas and power distribution, water management are clearly critical infrastructure, the notion of what is critical is growing within the digital landscape. For example; where only a couple of years ago having no Internet connectivity was an inconvenience, nowadays it is a major obstacle, people get lost without their route navigation software, etc.

A major boost for the media attention for SCADA security was the discovery 'in the wild' of a very advanced targeted virus; *Stuxnet*. Stuxnet is a very sophisticated piece of malware, combining advanced techniques, some of which are already introduced above. It uses several zero day vulnerabilities, is aware of virus scanners that might be able to detect it and adapts itself to avoid them, uses techniques to remain hidden and is able to update itself through a command and control center. A stolen certificate is used to pretend to be a valid driver, so it will be installed without prompting the user.

Current anti-virus measures are not able to deal with sophisticated attacks such as stuxnet. Advanced persistent threats and digital warfare continue to grow. New protection methods will be needed to defend against such targeted digital attacks. [5]

8.3 Defending against network security threats

Above we have already mentioned several methods of countering specific network threats. Here we treat three general categories of network defense technologies: Virus scanners, Network Intrusion Detection Systems (IDS, NIDS), and firewalls. Virus scanners try to detect and disable or remove (clean) malware, intrusion detection systems try to detect malicious traffic and firewalls try to block malicious traffic.

A key problem that we see returning in each of these technologies, as well as many other security mechanisms, (input filtering, access control, biometrics, etc.) is the need to distinguish 'good' cases (code, traffic, input, requests, measurements, etc.) from 'bad' cases. In general we have two ways of approaching this:

[4] `www.nytimes.com/2011/10/15/world/europe/uproar-in-germany-on-police-use-of-surveillance-software.html`, `jurist.org/thisday/2012/02/german-high-court-rejected-police-computer-hacking-efforts.php`, etc.

[5] At the TU/e we have several projects devoted to this topic, for example the SpySpot project `security1.win.tue.nl/spyspot/`.

- *Whitelisting*; by using an access control matrix to describe what traffic (or …) is 'good'. Anything else is considered 'bad'.
- *Blacklisting*; by describing what traffic is 'bad'. Any other traffic is considered 'good'. Many virus scanners work in this way; they look for (patterns of) known viruses in programs.

The AC matrix tries to simply list all allowed ('good') cases. As we have seen this quickly becomes unmanageable, thus prompting the introduction of more advanced specification methods such as RBAC and XACML. In other settings there are simply way too many possibilities to create a simple list. We thus need to express a model for 'good' or 'bad' in a more efficient way. For example input filtering against SQL is a form of blacklisting where we use a set of rules, for example, "the ' symbol is not allowed in a user input used to construct an SQL query". or replace the offending character with a safe encoding.)

Anti virus Anti virus software tries to identify malware and prevent it from causing harm. Typically by periodically scanning the whole system, checking content as it comes in (e.g. downloads in the browser, incoming email) and on access scan; programs get scanned on the fly as they are started. It recognizes malware in different ways; *signature based* recognition compares the scanned content against a list of known viruses. This requires constant updating of the list of 'fingerprints' of known viruses. Viruses not on the list yet will be missed. Also variations of the same virus may be missed if the characteristic fingerprint is changed or masked.

The number of viruses is huge and growing quickly. Though virus software uses several mechanisms to reduce the performance cost, such as checking for changes (storing e.g. a CRC of a scanned program) and not rechecking unchanged programs, purely signature based detection becomes unfeasible.

Heuristic recognition tries to recognize malware for example by running the program in isolation (i.e. a 'sandbox') to examine its behaviour or decompiling to look whether the code contains characteristic malware patterns. It is not possible in general to decide whether content is 'malicious' thus we can only approximate while making mistakes and mislabeling some content.

There are all clearly signature/rule based blacklisting approaches; they try to track and recognize the 'bad' cases. There is some problem with *false positives*; some legitimate programs will match a signature or heuristic rule. The *false negative* problem, however, is much bigger; previously unknown malware cannot be detected. This is a general issue for techniques using signature or rule based models; they cannot deal with new cases which were not considered when the signatures/rules were made.

There are also whitelisting approaches that try to combat malware. For example, Windows will normally only install device drivers that are digitally signed. This is thus a rule based whitelist; "accept only those drivers that are signed by the correct authority". The authority is trusted to check that the code is not malicious. Reputation based systems look at how long has the program been around, how often is it used, where and in which context, based on feedback from millions of machines (note the potential privacy risk). Here the community at large instead of a single authority is used as a source of trust in the program.

Anti-anti-virus Polymorphic viruses, the 'retro-viruses' of the digital world, try to prevent detection by anti-virus software by mutating, on occasion changing its appearance in new copies. In this way a new signature may no longer be valid. One way of mutating is to encrypt the payload with a (new randomly generated) key (the key used will have to be included in the code to be able to decrypt but a basic pattern analysis will not reveal this). By not mutating too often it also makes the job for analysts harder (i.e. by not providing many different copies of the same malware).

Stealth techniques employed by viruses to hide from virus scanners include intercepting read requests and replacing reads of infected files by clean copies. Related to this is rootkits; software which operates 'at the root of the system'; it is not visible at the operating system level. Virus protection that uses OS calls will not be able to detect such viruses.

Other viruses take a more active approach actually attacking the anti-virus software; stopping the processes belonging to such software, making anti-virus (update) sites unreachable (e.g. by altering DNS settings).

Thus while it is necessary to run anti-virus software to catch the many basic attacks, more sophisticated or simply just new attacks are likely to escape detection.

IDS Malware will often use the network to spread or otherwise execute malicious activities (for example botnet activities such as sending loads of spam mail). Intrusion detection systems (IDS) monitor the network to find (and possibly block) malicious activities. IDS systems come in two main classes; *signature* and *anomaly* based. As mentioned, signature based systems check for patterns in the traffic that match known attacks. Anomaly based systems look for 'abnormal' behavior; anything that is not the usual behavior on the network could indicate an attack. These classes thus have the same drawbacks as observed for virus scanners.

Not all attacks can be detected and not all detected anomalies (or signature matches) are attacks. A way to measure the quality of an IDS is to look at its false negative rate (missed malicious traffic) versus its false positive (false alarm) rate. Perfect detection rate with no false negatives and no positives is impossible. Instead we are left with a trade-off between detection rate and the amount of (false) alarms that are raised and need to dealt with by the security officer.

Signature based systems have a high false negative rate on new attacks while anomaly based systems will suffer from false positives.

In signature based systems the trade-off is set by how 'wide' the rules are (with 'everything is an attack' as an extreme). With anomaly based systems it is set by choosing a threshold on how uncommon an activity has to be to be called an anomaly (such as 'traffic to addresses that occurred less than t percent of the cases during the training is anomalous'.)

Firewalls We already briefly mentioned firewalls when taking about IP spoofing. As we have seen above, the effect of different attacks and the ease with which they are employed depends on the capabilities of an attacker. A local attacker is able to do much more easily cause more damage then a remote attacker. As such it makes sense to try to build barriers between different parts

of the network. Firewalls do exactly this by filtering the traffic between two networks, e.g. an organization's LAN and the Internet, an organization's web services and its intranet, a single PC and the intranet, etc.

Filtering can happen at different layers of abstraction (and thus at different network layers). For example a basic packet filtering firewall (working at the network layer) can help against IP spoofing of local addresses by outside attackers and can block access to ports (and services) that should not be accessible. If working at the TCP level, it will likely need to remember which sessions are open to be able to distinguish real responses from spoofed messages; a *stateful firewall*. Increased filter complexity can have a significant impact on the resources needed and thus the performance of the firewall. Going up to the application layer one can try to block dangerous data or known threats; such as remove active components from web pages, macros from word documents, block downloaded files containing viruses, tag spam and phishing emails, etc. Of course this greatly increases the complexity of the firewall; instead of looking at single packets one needs to understand the protocol being used, extract and reconstruct the data being sent, interpret and evaluate the data to determine whether it is harmful.

A main difference between firewalls and intrusion detection systems is that the former actually blocks traffic considered to be malicious. With an IDS the operator still needs to respond to deal with the attack (for example by defining a new firewall rule.) The use of firewalls thus has a direct impact on availability and usability; if we block traffic then availability and usability will go down. For example, by blocking port 22 of all machines except the public SSH server you help protect the network (e.g. a mis-configured machine vulnerable on port 22 would not be accessible from the outside) but also disallow other computers on the network from offering SSH connections. Also, the firewall will impact network performance with more advanced filtering requiring more effort thus adding to cost and/or leading to further loss of performance.

A good policy (a model of 'good' and/or 'bad' traffic) is thus needed to make firewalls useful; one that implements optimal trade-offs between protecting against network risks and keeping network services available. The risks, ways to detect attacks and network services needed change dynamically, making maintaining an effective firewall challenging.

The use of anomaly based approaches is uncommon when actually blocking traffic; not only is there the risk of a high false positive rate but also the reason why a particular message did not arrive might become unclear. Knowing a set of rules for the firewall a user can understand why certain connections do not work and what to change in the rules to make them work. However, understanding a machine-learned model of traffic used for anomaly detection, let alone updating it, is a lot more difficult.

Note that the firewall can only inspect the traffic that it can see. If data is encrypted (a best practice for securing a connection) this limits the possibilities for the firewall.

Summarizing, a firewall is a very useful tool and is employed by nearly anyone operating a network. They are even built into operating systems to protect the (network consisting of only the) PC from the network it is connected to. Still, given its inherent limitations it is by itself not sufficient to protect a local network. Thus it is typically combined with IDS to find threats on the local network and anti-virus on the end-points; the machines on the network.

Data gathering and training Anomaly based whitelisting approaches need normal non-malicious traffic for training; i.e. for creating a model of normal traffic. Simply monitoring the network for a sufficient period of time is a way to gather such traffic. However, what is sufficient may not be obvious; all normal traffic has to occur sufficiently often. Also, there may already be attacks in progress thus we have to be careful not to consider these as normal behaviour as well.

Signature/rule based blacklisting requires that one knows the attacks that are likely to be used against the system being protected. But how do we know what the attacks are and what type of traffic should be blocked? One way is to use fake systems or networks that collect data on attacks. Attackers are lured to these systems and monitored to reveal what vulnerabilities in which services they try to exploit. Sharing this information with others will allow making more complete lists. The trade-off is that it will also tell the attackers what attacks are known and thus help them to craft attacks that will not be detected (or apply known attacks on not-up-to-date systems).

8.4 Summary

In this chapter we have looked at Network security and Web service security. We first looked at threats at the different network layers and specifically application layer threats for web servers. Firewalls to try to keep attacker from entering the local network, intrusion detection systems to find an attacker that managed to enter the local network and anti-virus solutions to protect against attacks that manage to reach the end-points. We identify black/white listing and signature/anomaly based models as methods used by these approaches to classify traffic/content into normal and malicious.

8.4.1 Literature

Suggested reading

- Sections 8.6-8.10 of the book Computer Networking: A Top-Down Approach [20, Sec. 8.6-8.10],
- Hunting the Unknown, White-Box Database Leakage Detection [12].

8.5 Homework Exercises

1. Give a threat for each of the four layers in the simplified OSI model.
2. Give two threats, for different translations between layers.
3. In the following lists which is the odd man out and why[6]:
    - Session hijacking, SQL injection, XSRF and XSS.
    - DNS spoofing, IP spoofing, MAC spoofing.
    - Eavesdropping, Honeypot, Stuxnet, Virus scanner, Worm.

[6]Different answers may be possible based on different reasons. Give what you consider to be the 'best' answers.

4. Once again consider the online music store scenario of Exercise 6 in Chapter 1. Enhance the requirements you extracted and security design that you made by considering threats and techniques introduces in this chapter. Indicate how you would apply the techniques, what trade-offs this imposes and what effect they have on both the attacks as well as on the goals of the legitimate actors in the system, and, if applicable what new goals/actors you need to introduce.

5. Figure 8.1 shows several example attacks and their place in the OSI stack. No example is given of an attack related to the translation between transport and network layer. Give a threat that you would consider to fit in this location.

6. Assume the script in Figure 8.11 is used on a website. What input could an attacker give (in the username and password fields) to:

   (a) login as user Alice.

   (b) test a guess (e.g. *MyPassword*) for Alice's password.

   (c) test whether Alice's password starts with *M*.

7. Consider the DNS poisoning attack against a local name server.

   (a) What is the attacker model; which implicit assumptions are made and why?

   (b) The attacker needs to get the guessed ID to match one of the IDs of one of the outgoing requests. Each ID is a (uniformly) randomly chosen 16-bit number. Suppose the attacker sends the same number of fake answers as there are requests. How many requests do there need to be for the attacker to have at least a 50% chance of succeeding? (Hint consider the change that none of the fake answers match any of the requests.)

8. The quality of an anomaly based IDS (or more generally the performance of any binary classifier) can be shown with a so called "Receiver operating characteristic (ROC) curve" which plots the false positive rate (false alerts) against the true positive (detection) rate (true alerts; equal to 1 - false negative rate). (Recall that setting a different threshold results in different trade-off between these two numbers.)

   (a) Draw a ROC curve with four classifiers; the first does random guessing, the second achieves a detection rate of 80% at the cost of a 40% false positive the third a detection rate of 90% at the cost of 30% false positive rate and finally a perfect classifier.

   (b) Assume we have a classifier that is below the line of the random guessing. How could we make a better classifier out of this?

Chapter 9

Cryptography

Cryptography is one of the main building blocks available to the network security engineer. In this chapter we describe the basics of *cryptography*, the design and building of *ciphers*, and *cryptanalysis*, as well as the analysis and breaking of ciphers. Together these are referred to as *cryptology*.

This chapter first discusses the security goals that one may want to achieve with cryptography and then treats the two main classes of cryptography: Symmetric and Public Key (also known as Asymmetric) Cryptography.

In our discussion we try to answer the following questions:

- How to defend against different attacker models using crypto?
- How to encrypt and decrypt large messages?
  - As encryption algorithms typically have a size limit on the messages that they encrypt we discuss how to encode larger messages split into multiple blocks.
- What does it mean to say "a cipher is secure"?
  - After giving some concrete examples of algorithms we look at how to evaluate the quality of a cryptographic algorithm.

9.1 Basics and Security Goals of Cryptography

If you want to communicate a message (e.g. over a network) and are concerned that someone may be listening in you could try to hide the message in such a way that the eavesdropper will not understand it.

The texts in Figure 9.1 both contain a hidden message - try to discover it. It may seem to be well hidden; at least it is not directly obvious. However, once the method is known it will no longer work (it relies on obscurity of the encryption method); once you know how to find the message in the first example the second will be trivial. The hidden message in the text in Figure 9.2 will likely also be quite easy to find.

The Kerckhoffs's principle[1] tells us that the security of an encryption technique should not rely on secrecy of the algorithm used. Instead, keeping a small

[1] after the Dutch cryptographer Auguste Kerckhoffs who stated this requirement in 1883.

Greetings to all at Oxford. Many thanks for your letter and for the summer examination package all Entry forms and Fess Forms should be ready for final dispatch to the Syndicate by Friday 20th or at the very latest, I'm told, by the 21st. Admin has improved here, though there's room for improvement still; just give us all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic A and O pattern. Certainly this sort of change, If implemented immediately would bring chaos.

Welcome back to Oxford. Thanks again, this letter explains the winter examination method and its related forms. Early submission does guarantee full and early feedback but does not influence the grading of the quality of the work done. A full grade report will be available once the deadline for submissions has passed. In it the evaluation is explained. The evaluation is final as the criteria for the work are now known.

Figure 9.1: Two texts with a hidden message

A final greeting to our Oxford graduates. Though with a slight delay, we hope this letter finds you well. The new variation in the forms attached shows how our alumni will continue to play a key role in our school and will not be forgotten. Instead we hope that you continue to work with us, and any contribution that you can bring, either directly or indirectly, will be appreciated.

Figure 9.2: Another hidden message

piece of input information (the *key*) secret should be sufficient. Note that this is an instance of the more general 'not relying on security through obscurity' principle.

Figure 9.1 shows the basic operation of a cryptography system; an encryption system protects a message by creating a jumbled version of the message using a known method (the encryption algorithm) and a small piece of secret information (the encryption key). With a corresponding secret (the decryption key) and another known method (the decryption algorithm) the secret message can be recovered. Trying to use the decryption method without knowing the correct key provides no information about the secret message.

Encrypting data thus aims to protect its *confidentiality*. There are also cryptographic techniques such as digital signatures (see also Section 9.3), which allow checking *integrity*. *Authenticity* and providing *non-repudiation* can also be achieved using encryption schemes.

Several *privacy* enhancing techniques use techniques similar to asymmetric cryptography and indirectly, through confidentiality, cryptography can contribute to privacy. However, privacy is typically not a direct goal of cryptography; your data is under control of other entities which have control over the data. Privacy requirements restrict what they may do with the data, not the ability to get to the data.

Encryption clearly has a negative impact on *availability*. Decryption aims to maintain availability in the presence of encryption, of course, only for those with the correct decryption keys. Thus confidentiality and availability can be

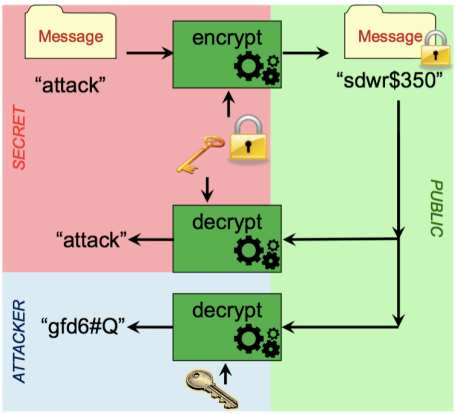


Figure 9.3: Basic overview of a crypto system

combined by ensuring only the correct parties that have the relevant decryption keys (recall the discussion on security requirements from the first chapter).

9.2 Symmetric Cryptography

The decryption needs a secret (the decryption key) related to the secret used for encryption (the encryption key). In symmetric cryptography the two secrets are actually the same; a single key is used both for the encryption and the decryption.

A symmetric key cipher system thus needs an *encryption algorithm*, a *decryption algorithm* and a *secret key*, which is used by both. If a message, referred to as a *plaintext*, is encrypted with a key and the resulting *cipher text* is later decrypted with the same key then the original plaintext should be recovered. Both the cipher text and decryption of the cipher text with a different (thus incorrect) key should make no sense to an attacker.

The only difference between a legitimate user and the attacker is that the legitimate user knows the key. The key should be chosen randomly to ensure that the attacker cannot simply guess it. In many symmetric key algorithms the key can be any bit-string of a given length and selecting a random string of this length is sufficient to generate a good key. However, for some ciphers, especially asymmetric ones, the keys have some structure and generating a good random key is more complex. A cipher should then also come with a probabilistic *key generation algorithm*.

With these notions in place we can now try to formally define what "makes no sense to an attacker" means. A possible and very strong definition is to require that there is no correlation with the plaintext at all. In this case, the security is *unconditional*; i.e. no matter the computational resources of an attacker, the attacker cannot learn anything new about the plaintext from the cipher text (or in other words - our attacker model assumes no limits on the computational resources the attacker has available).

Definition 9.2.0.1 (Unconditionally Secure) *We say a symmetric cipher with encryption Enc is unconditionally secure (also known as information theo-*

retically secure) if encryption with a random key gives a cipher text that is not correlated with the plaintext.

Formally, consider random variable $\mathbf{k}$ *(the randomly selected key). We require that for any cipher text* c *and potential plaintexts* p_1*,* p_2*:*

$$\mathbf{P}(c = Enc_{\mathbf{k}}(p_1)) = \mathbf{P}(c = Enc_{\mathbf{k}}(p_2))$$

where $\mathbf{P}$ *denotes the probability. That is, if one does not know the key each plaintext is equally likely to have been the one that was encrypted.*

An alternative, equivalent, definition is that for any probabilistic distribution over plaintexts (the attacker's a-priori estimate of how likely a given plaintext is) the attacker learns nothing new by obtaining the cipher text; the conditional probability of plaintexts given an encryption with a random key is the same as in the original distribution over plaintext.

Recall from Chapter 1 that a notion of 'secure' should consider both the goal to be reached (a security policy in terms of security attributes to be achieved) and under which conditions/situations this goal must be achieved (the attacker model). We can summarize these as follows: the term 'unconditionally secure' cipher refers to the security goal 'confidentiality of the plaintext' with security policy 'those who do not know the randomly generated key may not learn anything about the plaintext' and the attacker is one with unlimited computational resources, knows (Kerckhoff's principle) the cipher including the probability distribution of the keys (because the attacker knows the key generation algorithm) but does not know the actual key (outcomes of the random choices in the key generation algorithm are not known).

Unconditional security imposes a very strong requirement on the cipher. Is it actually possible to create a cipher which achieves this? This is indeed possible (assuming the number of possible plaintexts is finite) and can be achieved by a surprisingly simple system; the *one-time pad*.

plaintext bits	p_1	p_2	p_3	p_4	p_5	...
key bits	k_1	k_2	k_3	k_4	k_5	...
cipher text bits	$p_1 \oplus k_1$	$p_2 \oplus k_2$	$p_3 \oplus k_3$	$p_4 \oplus k_4$	$p_5 \oplus k_5$	...

Figure 9.4: One time pad

This system performs a bitwise exclusive or (XOR, $\oplus$) with a key which is as long as the plaintext and which can only be used for a single plaintext. Decryption is done by repeating this same operation; $(p \oplus k) \oplus k = p$ thus the plaintext bits are recovered in this way. Though simple and easy to understand, this system is rather impractical in use because it requires as many key bits as there are bits of messages to be encrypted. For communication this means that we first need to safely share a key which is as long as the message that we want to send. But if we have a secure way of sending a key then why not use this for the message directly? Thus the use is limited to cases where we can share a key in a way we cannot share the message itself - e.g. meet to share a key in advance, and later use this to enable communication over an insecure channel or share keys over a channel on which eavesdropping can be detected (as in quantum key distribution treated in the course 'Physical Aspects of Digital Security' (2IMS10) where security meets quantum theory).

Similarly if we use the one time pad for securely storing (rather than sending) some data we would need to securely store a key of equal length. Here the only remaining benefit is that an attacker would have to obtain both the key and encrypted data, which we could try to make difficult e.g. by storing them in different places. This is a form of so called secret sharing.

So the one time pad is secure[2] but needs impractically long keys. Could we not think of a more efficient system which achieves the same level of security? Unfortunately, such a system cannot exist (see Exercise 9). Thus people have been trying to create practical encryption systems which are 'good enough'. Below we look at some of these attempts.

Substitution Ciphers The Caesar cipher which, as its name suggests, was already used in Roman times, is a simple substitution cipher. Given a plaintext we simply shift each letter by a fixed number of places (e.g. A becomes D, B becomes E etc.) to obtain the cipher text. To decrypt we shift back by the same number of places.

plaintext letters	H	E	L	L	O	...
key letter (repeats)	C	C	C	C	C	...
cipher text letters	K	H	O	O	R	...

Figure 9.5: Caesar cipher with key = C (+3)

It is clear that this cipher does not offer much security; there are only 26 possible key values (of which one is not really useful; consider which one and why) so we can only hope for $\log_2 26$ (i.e. less than 5) bits of security at best. *Conclusion* 1: We need a sufficiently large key space.

However, the Ceasar cipher can be broken even more easily. The patterns in the text remain unaltered (e.g. in Figure 9.5 the double L becomes double O) allowing knowledge of text patterns to be used to easily find the key. For example, by looking at the frequency of the letters one can conclude which letter is most likely encryption of the letter E which also directly gives the key used.

The Caesar cipher uses a very structured (linear) operation to encode letters; a rotation in which each letter moves by the same amount. If instead as substitution we would use an arbitrary 1-to-1 mapping from letters to letters, (also called bijection or permutation) where e.g. A could be encoded as E, B as Z, C as P, etc. The key is the mapping itself which we could store as a sequence of 26 letters ($EZP\ldots$). The number of possible keys is now 26! or approximately 2^{88} possibilities, which corresponds to 88 bits. This may seem a reassuring number as trying all keys, say at a million billion tries per second, would still take thousands of years. Yet breaking such a cipher can be done, even by hand, by looking at patterns in the text rather than trying all possible keys (there is actually a type of puzzle based on this idea; in a filled in crossword each letter is replaced by a number and the goal is to find the corresponding mapping between numbers and letters). *Conclusion* 2: The non-linearity of the substitution is a strength but not sufficient when working with single letters. Also storage requirements for keys need to be considered.

[2] What did that mean again in this setting ...

Thus substitution of a single letter at a time is insufficient. What if we increase the block size, i.e. always encrypt a number of letters at the same time. If we apply this principle to the Caesar cipher and thus use a word instead of a letter as the key we obtain the Vigenere cipher. (The Caesar cipher is a Vigenere cipher with a blocksize 1).

plaintext letters	H	E	L	L	O	...
key word (repeats)	B	Y	E	B	Y	...
cipher text letters	J	D	Q	N	N	...

Figure 9.6: Vigenere cipher with key=BYE

With a block size of n letters the size of the key space becomes 26^n. However, again an attacker can mount a more effective attack than just trying all possible keys; first the attacker guesses the length of the key n. This guess can be verified e.g. by checking that the frequencies of each n-th letter show the same pattern as the frequencies normally occurring in a text or simply by the fact that the next step succeeds. Finding the key is then an easy exercise of attacking a Ceasar cipher n times.

Thus simply increasing the block size does not, by itself, solve the problem. When our message is larger than the block size the repetitions still allow recognition of patterns. (What if our message is not longer than the block size and we only send one message so no repetitions occur?[3]) *Conclusion* 3: Just increasing the block size helps too little if the attacker can solve the problem one part at a time.

From the conclusions above a natural goal would be to combine larger block sizes with non-linear operations. Note, however, that a random 1-to-1 function of the entire block (for example with block size 3 a mapping could be: AAA maps to EQN, AAB maps to XPA, ...ZZZ maps to SDI), while hard to break, would give prohibitively large keys ($EQN, XPA, \ldots, ZZZ$ is over 50K letters long). *Conclusion* 4: We would like to have random 1-to-1 mapping without needing to store all its outputs. We could try to simulate this by not really using a random function but one that we can compute based on a small piece of random data that then is our key. Of course the output should look random; it should not depend on the input or the key in an easy to detect way (such as a linear relation). Modern block ciphers use this principle but before we can describe those we need one more ingredient.

Transposition Ciphers Instead of changing the letters in a text we could also try to hide the text by changing the order of the letters, i.e. transposing the text; applying a permutation to the text. This in itself, however, is easy to detect: the frequency of letters does not change so it may be easy to recognize. Also, if a fixed permutation is used there is no key. Hence once the method is known the security is lost. If instead a random permutation is used we have the problem of how to effectively store what permutation we are using. *Conclusion 5*: Transposition tries to hide which part of the output describes which part of the input but by itself is insufficient.

[3]Then it becomes equivalent to the one time pad just using letters instead of bits.

Modern Block Ciphers From the discussion above we can conclude that we want to:

- have sufficiently large keys (Conclusion 1) but not too large (Conclusion 2). If the best an attacker can do is to try all keys then 128 or 256 bits are commonly used sizes. The key space grows exponentially in the size of the key; if everybody in the world had a billion processors each able to try a billion keys per second it would still take over a thousand years to try 2^{128} possibilities while 2^{256} approaches the estimated number of atoms in the observable universe.
- work with sufficiently large blocks at a time (recall Conclusion 2) and,
- have output depend on all input (recall Conclusion 3); we don't want the attacker to be able to analyse the cipher text one piece at a time. We can use transposition to help achieve this but by itself it will not be enough (recall Conclusion 5).
- prevent the attacker from detecting how the output depends on the input or on the key (recall Conclusion 4).

By mixing substitutions (confusion) with permutations (diffusion) one can try to achieve these goals and simulate an encryption which is a large random permutation of all plaintext.

Let us design an encryption method: - we use blocks of input together (for example 128 bits) - we use a sufficiently large random key (for example also 128 bits) - we transform the input in a way that seems random:

Step 1 We try to hide the input. We can do this for example by the method used in the one time pad and in Vigenere cipher (for letters); we add the key to the input one bit at a time (that is we XOR the input and the key). Note that (so far) an attacker could analyze the result one bit at a time and the relation of output with input and key is linear. We thus need to mix the bits and hide this relation in the next steps.

Step 2: Confusion (Substitution) We take a small piece of the result (such as 1 byte) and put it through a fixed 1-to-1 mapping. This fixed 1-to-1 mapping is typically called an **S-box** (substitution-box). The S-box brings a non-linear element to the encryption but we have to keep it small and fixed to prevent having to store too much data. However, it is still clear which byte belongs to which input byte. As a result, the attacker now has to analyze a byte at a time instead of a bit at a time.

Step 3: Diffusion (Permutation) Mix the bits by moving them around (permutation) and possibly doing a (linear) combination. By moving the bits we try to hide how output bits depend on input bits so the attacker will not know which bits belong to which byte.

Of course we need to prevent the attacker from simply reversing step 3. We do this by hiding and combining the output bits again; i.e. we repeat steps 1 to 3 (i.e. go to round 2). The second round of step 2 will again combine 8 bits but, because of step 3, each of these bits come from different outputs of round 1.

Thus each in turn depends on a byte (due to step 2 in round 1) of the input and the key. By repeating steps 1-3 several times (rounds) we ensure the attacker will have to analyze the whole block at once and in a non-trivial way (due to the included non-linear steps).

The above sketches the main idea of many of the modern block ciphers (substitution-permutation ciphers). Each use repeated rounds applying confusion and diffusion though they differ in the order of the steps and how they mix the key with the input. Shuffling key bits between rounds, resulting in different 'round keys', is also common. In Section 9.5 we will see two examples of this in some more detail; the *Data Encryption Standard* and its successor, the *Advanced Encryption Standard*. Now we first move to the basics of asymmetric (a.k.a. public key) cryptography.

9.3 Public Key Cryptography

Suppose that Alice wants to be able to securely communicate with a lot of different people without any of them being able to eavesdrop on communications with the others. She could use the symmetric cryptography techniques we discussed above to achieve this. However, Alice will need to share a *different* key with each of these people and will have to store all of these keys securely.[4]

A way to solve this problem is to use asymmetric cryptography. An asymmetric system uses different keys to encrypt and decrypt data. Using such a system Alice could have a *public key*[5] that she could share with everyone and that others can use to encrypt messages meant for Alice so only she can decrypt it with her *private key* that she keeps secret. If Alice wants to send a message herself, e.g. to Bob, she uses his public key to encrypt it.

One can compare a public key system to a padlock; anyone who has the padlock (public key) can lock it (encrypt a message) but only the one with the (private) key can unlock (decrypt) it. A symmetric system on the other hand is like a lockbox where you need the same key both to put things in and take things out.

So how does this help; Alice still needs a different (public) key of everyone she wants to send messages to in addition to her private key? The advantage is that, while in a symmetric system she has to keep all keys confidential, with the public key system this is only needed for her own private key. (In both systems we will need to protect the integrity of all keys.)

For the public key approach above to work we need to satisfy the following:

- It should not be possible to derive information about encrypted messages or the secret key from the public key.
- Alice needs to be sure the 'public key of Bob' really belongs to Bob.

The need for the first property is quite obvious but less obvious is how we can achieve this; not only do we have to keep information secret but we have to keep it secret from somebody who has 'half of the answer' (the public key). The second requirement is related to key management; though Alice will not need to keep the public key of Bob secret, she does need to make sure it is correct and not the key of say Mallory.

[4] You are of course thinking: what exactly does 'securely' mean here?

[5] Hence asymmetric cryptography is also referred to as public key cryptography.

Authenticity of messages. The discussion so far focusses on achieving confidentiality of messages. Let's, for a moment, consider the other security attributes that we may want to achieve. Suppose we wish to achieve authenticity (and with it integrity) rather than confidentiality. With a symmetric key system Alice knows that if she decrypts a message and it makes sense, then it must have been created by Bob (or by herself); no one else has the key so no one can make cipher texts that decrypt to meaningful plaintexts. With public key cryptography we can also achieve authenticity but we have to 'turn the public key cryptography schema around'; Alice *signs* a message with her private key and everybody is able to check this *digital signature* using her public key. For some ciphers, such as RSA (see Section 9.5), this 'turning around' can be achieved by simply using decryption for signing and encryption for checking a signature.

Note that we could also use digital signatures to protect the integrity of public keys. For example, if Alice signs 'Bob's key is 2ef8d83441e...' and stores this she can then tell if someone has tampered with the stored keys as the signature will no longer be correct. Changing both the stored key and the signature is not an option for an attacker as only Alice can make valid signatures. This is an example of a *certificate* which we shortly mentioned in Chapter 8). To give some idea of the mathematics that make public key schemas, including digital signatures, possible we now look at the Diffie-Hellman key exchange protocol.

Diffie-Hellman Key Exchange Consider the following setup: Alice and Bob want to create a shared key, e.g. to be able to use a symmetric cipher, but are worried that someone may overhear their communication. Diffie-Hellman Key Exchange could be used to solve this. In this system two public parameters are set: a large prime number p and a *generator* $a < p$. Alice and Bob both generate a random number, x and y respectively, raise a to the power of this random number and send this to the other. The number r they receive they again raise to the power of their random number and the result is the shared secret key. Alice and Bob share the same key.
Alice uses
$r_{bob}^x \mod p = (a^y \mod p)^x \mod p = a^{yx} \mod p$,
and Bob uses
$r_{alice}^y \mod p = (a^x \mod p)^y \mod p = a^{xy} \mod p$.

An eavesdropper could obtain r_a and r_b. However to make the key e.g. from r_{alice} one would need y. The attacker could try to obtain y from r_b because $r_{bob} = a^y \mod p$ in which only y is unknown. However, this is an instance of solving a *discrete logarithm* problem in a finite group for which no efficient algorithms are known. Thus if we use a large prime number p an eavesdropper is highly unlikely to obtain the shared secret key.

This is a characteristic property of asymmetric ciphers; the security argument is that the attacker even if she knows the public key and the cipher text would have to solve a 'hard problem' to extract the secret key and/or the plaintext.

Note that the attacker model is very important for the security of this scheme. For example, the scheme is not secure against a *man in the middle* attack. In the man in the middle attacker model the attacker is able to not only listen to messages but actually intercept and change them. If Eve can alter messages

sent by Alice and Bob she could replace a^x and a^y by $a^{x'}$ and $a^{y'}$ after which both Alice ($a^{xy'}$) and Bob ($a^{x'y}$) share keys with Eve while thinking they share a key with each other.

9.4 Block modes: Encrypting more than one block

Block ciphers take a fixed size (64 bits, 128 and 256 bits are common sizes) block to encrypt. Asymmetric cryptography typically takes an element of a group of size n, thus encoding at most $\log_2(n)$ bits (e.g. 256, 512, 1.024 and 2.048 are common sizes for n in bits). Thus, to encode large messages, we will need to split the message into multiple blocks.

We can divide our message up into blocks of the correct size and simply treat each block separately. This approach, called Electronic Code Book (ECB) mode, has several drawbacks. For example; the same block will encrypt to the same cipher text every time, thus patterns in the original text will still be visible in the encrypted version of the text. (This problem also occurred for the Caesar/Vigener ciphers but now relates to blocks instead of letters/words).

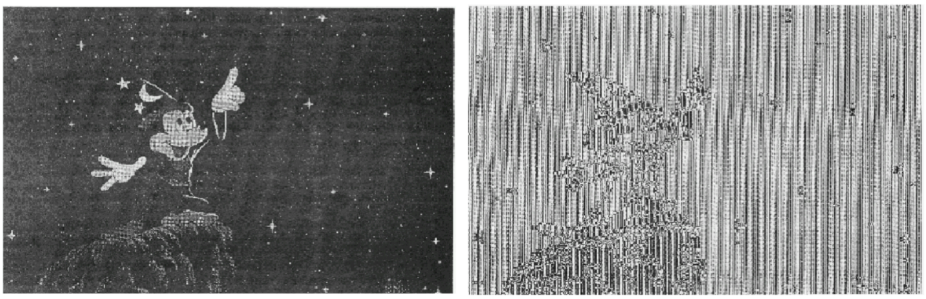


Figure 9.7: Encryption using ECB mode

The effect is evident when encrypting an image with a symmetric cipher in ECB mode as in Figure 9.7. If we simply treat the encrypted data as an image it is garbled but the original image can still be recognized. A clear case of 'Mickey Mouse encryption' one could say.

Instead of treating blocks separately, one could combine a block to be encrypted with the previous blocks (and/or their encryption) in some way. Then the encryption of the block becomes dependent on the previous block(s) and any structure in the plaintext is hidden. One way to do this is by using Cipher Block Chaining (CBC) mode. This mode first XORs the previous encrypted block with the plaintext before encrypting it. For the first block, where no previous encrypted block is available, an 'initialization vector' is used. With CBC the data that is encrypted is basically random, ensuring no structure remains that could be recognized in the cipher text blocks. In Figure 9.8 we encrypt the same picture as in Figure 9.7 but now with CBC mode and no discernable pattern remains.

Two other modes are Cipher Feedback (CFB) mode and Output Feedback (OFB) mode. The OFB mode turns a block cipher into a *stream cipher*. In a stream cipher a pseudorandom stream of key bits is generated from the shared

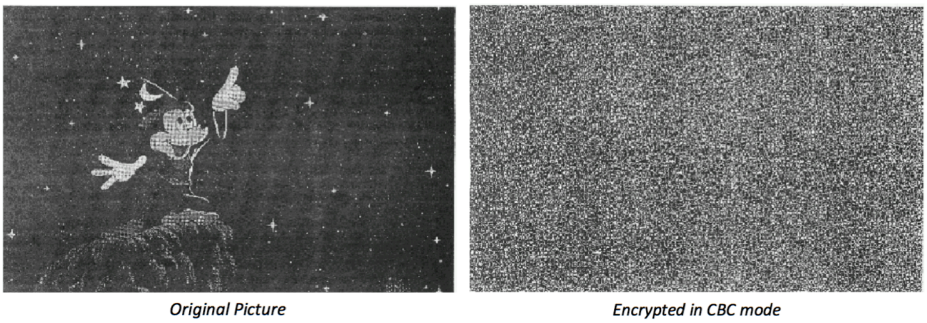


Figure 9.8: Encryption using CBC mode

symmetric key and this key stream is XORed with the plaintext to obtain the cipher text. Note the similarity with the one-time pad; if the key stream is 'sufficiently random' then so is the cipher text. Also, like with the one-time pad, the same key(stream) should not be reused. With OFB mode, an initialization vector is encrypted to give the first block of key bits which then also replaces the initialization vector and is re-encrypted to form the next block of key bits, etc. (Using a different initialization vector enables one to generate multiple streams from the same key in OFB mode.) The CFB mode is a slight variation of CBC with a structure that resembles a stream cipher. In CBC an initialization vector is encrypted to get the first block of random bits and the resulting cipher text is then re-encrypted, thus re-seeding the pseudorandom number stream, but here with data which also depends on the plain text.

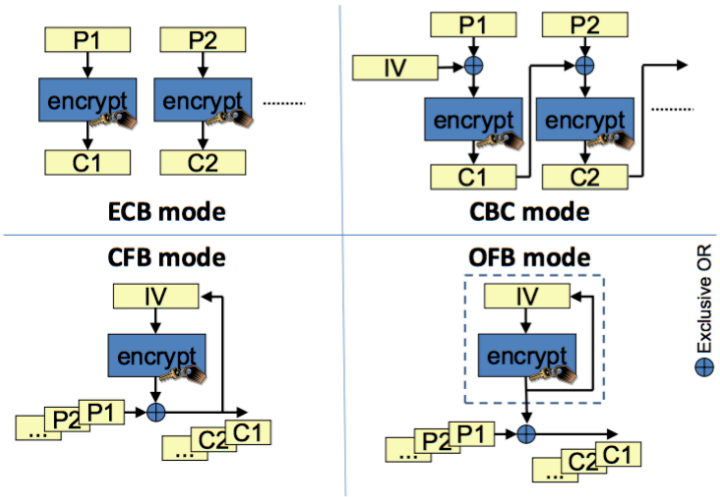


Figure 9.9: Overview of Block modes ECB, CBC, CFB and OFB

9.5 Example Algorithms

To get a better idea of the inner workings of algorithms we now look at some common algorithms.

9.5.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) was adopted by the US National Bureau of Standards (NBS), now named National Institute of Standards and Technology (NIST), in 1976. The key used by DES is effectively 56 bits long. (It specifies 64 bit keys but 8 of these are parity bits.) DES uses a 16 rounds *Feistel network* structure (see Figure 9.10).

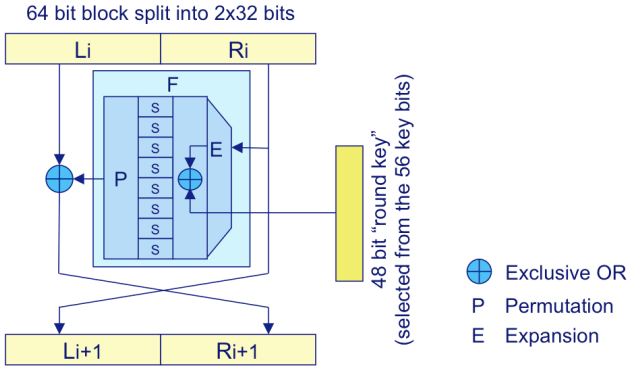


Figure 9.10: In the Feistel structure a round consists of splitting the input into two parts, applying a function F to the right hand side and XOR-ing the result with the left hand side. This gives the new right hand side. The new left hand side is the old right hand side.

In DES the function F works as follows: A simple key schedule determines which of the 56 bits of the key form the 48 bits of the round key for each round. The input (R_i) is extended (by repeating some bits) to 48 bits and the result XOR-ed with the round key. The eight different DES S-boxes, each taking 6-bits of input and producing 4 bits of output are then applied yielding 32 bits of output. The final step in computing F is permuting these 32 bits.

One of the advantages of the Feistel network structure is that the decryption is very similar to the encryption: undoing a single round is exactly the same just with left and right parts swapped. (Check this for yourself; apply a round with key K_i and input R_{i+1}, L_{i+1} to see it gives as output R_i, L_i.) Note that the function F does not have to be invertible. The DES S-boxes, in particular, are clearly not invertible as they have less bits of output than input.

Fear that the influence of NSA on the design introduced specific weaknesses that they could exploit was later countered by the relatively good resistance of the cipher to 'differential cryptanalysis', a technique published in the late 1980s but apparently already known to the NSA and the IBM designers at the time DES was made.

A main issue with DES is its limited key size. While, when DES was initially published, it was considered that a machine that would break this cipher

in a reasonable amount of time would be unrealistically expensive, already in 1977 a theoretical 20M dollar machine which could break keys in a day was described [14]. In 1998 the Electronic Frontier Foundation (EFF) actually built Deep Crack for less than $250,000 which can crack a key in a few days [1]. Currently 'brute-force' techniques for cryptanalysis not only benefit from individually faster machines but also from massive parallelism such as distributed computing and the use of graphics cards.

In short DES cannot be considered secure[6] anymore. As a solution to the short key, *triple DES* was introduced. As the name suggests triple DES uses three DES operations with a set of three 56 bit keys; two DES encryptions with a DES decryption in between. [7]

By choosing all three keys different we get an encryption scheme with a 168 bit key. It is also possible to take the first and second key different but the third key the same as the first key. This gives an encryption scheme with a 112 bit key. Of course the key length alone does not determine how much security a cipher really offers. Estimates of security offered against the best known attacks are 112 bits for the first and 80 bits for the second scheme. (The detailed cryptanalysis of this method are beyond the scope of this course.) Recall that 'n-bits of security' reflects that using the best known attack the attacker would have to do a similar amount of work as randomly trying to guess an n-bit key.

9.5.2 Advanced Encryption Standard (AES)

A new standard was introduced in 2001: the Advanced Encryption Standard (AES) based on the Rijndael cipher. This cipher was chosen from several competing encryption schemes (the 'AES candidates'). AES uses a 128, 192 or 256 bit key to encrypt 128bit blocks. Here we will focus on the 128 bits version of the algorithm.

The input to AES is ordered into a 4 x 4 matrix of bytes and bytes are seen as elements of finite field GF(2^8). The following four basic operations are used in AES:

AddRoundKey which XORs the current state with the round key. The round key is also represented by a 4x4 byte matrix. We do not treat how the round key is derived from the main key here.

SubBytes which applies an S-box substitution on each byte of the state. The S-box that is used is given below (in hexadecimal format). Unlike the DES S-box the AES S-box is invertible. Also, the AES S-box can be expressed as a combination of several functions, allowing it to be computed rather than stored, which may be relevant for devices with very limited storage capability.

ShiftRows which rotates each row a number of steps. Row n is rotated $n-1$ places to the left, i.e. the first row is not changed, the second row is rotated

[6] Remember to consider what 'secure' means in this context.

[7] Choosing the first two keys of triple DES equal the decryption cancels out the first encryption and the end result is the same as doing a single DES encryption with the third key. In this way we get backward compatibility with single DES; triple DES encryption hardware can also be used to do DES encryptions.

```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75
40 |09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
50 |53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
60 |d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8
70 |51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
80 |cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
90 |60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db
a0 |e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79
b0 |e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08
c0 |ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
d0 |70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e
e0 |e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
f0 |8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16
```

Figure 9.11: The AES S-box

1 place to the left etc. As an effect, each column after the shift depends on every column before the shift.

MixColumns which combines the four bytes in each column to a new column by multiplying with the following matrix:

$$\begin{matrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{matrix}$$

Within GF(2^8) addition and multiplication work as follows: Addition is XOR. Multiplication by 2 is a shift 1-bit to the left. If the most significant bit is 1 this results in a result larger than 8 bits so the modulus operation in GF(2^8) is applied which corresponds to doing an XOR with 1B (hexadecimal), such that the result fits in 8 bits again. MixColumns can be inverted by multiplying with a different matrix.

The basic functions are combined into ten rounds as follows: In the first round a single AddRoundKey is performed. Rounds two to nine perform the sequence SubBytes, ShiftRows, MixColumns, AddRoundKey and the final round leaves out MixColumns, thus performing SubBytes, ShiftRows, AddRoundKey.

Each AES operation is invertible. Decryption simply inverts each encryption step in the reverse order.

9.5.3 RSA

In 1978 **R**ivest, **S**hamir and **A**dleman were the first to publish a public key system. Their RSA system works as follows:

Setup and key generation: Randomly choose two large prime numbers p, q and compute modulus $n = p * q$. Pick e and d such that $ed = 1 \mod \phi(n)$[8] (knowing p and q one can compute d for a given e e.g. by using the extended Euclidean algorithm). The public key is (e, n) and the private key (d, n). That is, e and modulus n are made public and d is kept secret. The way of choosing e and d guarantees that $a^{ed} = a \mod n$ for any a.

Encryption A plaintext can be any number less than n. To encrypt a plaintext P raise it to the power e (modulo n), $c = P^e \mod n$.

Decryption To decrypt a cipher text C raise it to the power d (modulo n), $P = C^d \mod n$.

The way of choosing e and d guarantees that $a^{ed} = a \mod n$ for any a. Thus decryption works because $C^d = P^{ed} \mod n = P \mod n$. The public information (e and n) is not sufficient to find d, p, q or $\phi(n)$ and this is essential as knowing any of these would allow (finding d and) decrypting messages.

The large prime numbers p and q are randomly chosen; an attacker should not be able to guess them. They should be sufficiently large as should their difference ($|p - q|$) to ensure factoring n is actually difficult. All users should have their own, distinct modulus n. The public key e is typically chosen to be convenient value such as 3, 7 or $2^{16} + 1$ as this allows for efficient encryption (and as we will give away this value anyway there is no need for it to be random). The private key d on the other hand should be big (close in size to n) to prevent it being guessed or derived efficiently from e and n.

As we are working modulo n there are up to n possible values. Thus if we work in blocks as with symmetric ciphers, the maximum block size with RSA is $\log_2(n)$. There are several ways to represent blocks as numbers in $0, \ldots, n$. Typically some form of padding is used, for example by forming the most significant bits from random *salt* creating a randomized encryption. The padding ensures that the numbers are not too small which would give several problems: if plaintext P is a small number then P^e may be smaller than n (recall that e may be small) and thus we are doing normal integer arithmetic where taking an eth-root is easy, instead of modular calculations where such an operation is hard.

Note that for RSA encrypting and decrypting are the same except for the use of a different key and, as $de = ed = 1 \mod \phi(n)$ we can reverse the role of the encryption and decryption key; i.e. thus first decrypting a message and then encrypting it also results in the same message. With this we can also use RSA to perform signing; by decrypting the message we want to sign we generate a signature that anyone can check with the public key and that only the holder of the secret key can create. (We typically sign hashes of message rather than messages themselves. If you are interested see the reading material, Handbook of applied cryptography [22, Chapter 1], for more on hashes.)

The structure of RSA encryption and decryption also gives several other properties, for example that the encryption of the product of two messages is the product of the encryptions (modulo n), $E(m * m', (e, n)) = E(m, (e, n)) * E(m', (e, n)) \mod n$. Combined with the observation above this allows one to create *blind signatures*, i.e. to have Alice sign Bob's message without Alice learning the message: Bob generates a random mask r, encrypt this mask with Alice's public key (e, n) and multiplies the message (modulo n) with the en-

[8] $\phi(n) = \phi(p * q) = (p - 1) * (q - 1)$ for primes p and q.

crypted mask. The result, $m * r^e$, is given to Alice to sign (decrypt). This text does not reveal anything about m to Alice and by signing it she creates the signature of m masked (i.e. multiplied) with r.

$$D(m * r^e, (d, n)) \;=\; (m * r^e)^d \mod n \;=\; m^d * r \mod n$$

Bob can now remove the mask by dividing by r.

9.6 Computational Security

RSA requires a significant amount of computation to encrypt and especially to decrypt. Also, the mathematical structure used for RSA implies that keys have to be big to prevent attackers from e.g. factoring the modulus n. So how large does the key need and what are the costs of working with such large numbers; how does the encryption and decryption performance compare to e.g. that of a block cipher?

Public key systems are typically significantly slower than symmetric systems and often require larger keys for the same level of security. When considering how large a key is needed, consider the capabilities of an attacker when they perform the attack. If the encrypted data is to stay secret for three decades then we need to plan ahead and consider the computational power that an attacker may have in 30 years.

If we use a large key, is an algorithm such as RSA secure? What does this exactly mean; when is RSA broken? In the symmetric setting we can achieve 'unconditional security' (i.e. information theoretical security) with the one-time pad. Even an attacker with unlimited computational power could not break this cipher because there was no way of validating whether a guessed/found key is correct. For RSA unconditional security is clearly not achievable; an attacker with unlimited computational power could theoretically factor modulus n for any size key which is enough to compute the private key from the public key. In the public key setting in general the attacker can always encrypt messages herself (as she knows the public key) and thus check whether these decrypt correctly (with a guessed secret key), allowing elimination of incorrect key guesses. Thus we cannot expect any unconditionally secure cipher in this setting.

Luckily no attacker has unlimited resources. Above we argued that breaking the system would require solving a problem that would be too hard. Thus we aim for *computational security*: breaking the cipher should not be computationally feasible for the attacker. But what exactly is hard and how can we be sure that the attacker cannot think of a way to break the system without solving this hard problem? To answer the first part of this question we recall some basic complexity theory.

Definition 9.6.0.1 *Consider a problem which can have instances of different sizes. We say such a problem is in complexity class $\mathcal{P}$ (polynomial time) if there exists a (deterministic) algorithm which solves the problem and the time it takes grows at most polynomial in the size of the problem.*

We say a problem is in complexity class $\mathcal{NP}$ if there exists a non-deterministic algorithm which solves *the problem in polynomial time. (Equivalently we could say that there exists an algorithm that can* ***check*** *a given solution to the problem in at most polynomial time.)*

We say that a problem is 'hard' when it is not in $\mathcal{P}$. Having a hard problem, however, is not enough to construct a public key cryptography system; we need a problem that is hard for an attacker but easy for the holder of the private key. We need a *trapdoor* function, i.e. a function that is:

- easy to compute in one direction (to be able to encrypt)
- hard to compute the inverse without the secret (an attacker should not be able to decrypt)
- easy to compute the inverse with the secret (the secret key holder should be able to decrypt)

A first natural question to ask is: do such functions actually exist? The perhaps surprising answer is that this is not known. A fundamental unanswered question of complexity theory is whether $\mathcal{P}$ is equal to $\mathcal{NP}$. Suppose that we were to find a trapdoor function then computing its inverse (by the attacker) is a problem that is not in $\mathcal{P}$ but is in $\mathcal{NP}$ which would prove that $\mathcal{P} \neq \mathcal{NP}$: computing the inverse is not allowed to be in $\mathcal{P}$ as it must be a hard problem for the attacker. Yet, computing this inverse is easy for the holder of the key. Thus a non-deterministic algorithm could first 'guess' the secret and then perform the same computations as the key holder.

Still, though no-one has proven the existence of trapdoor functions, many candidates exist which have been studied extensively and for which no efficient algorithm to compute the inverse are known. One example is factoring the product of two primes; creating the product from the primes is easy while factoring a (large) number is difficult. However, if you know one of the two primes finding the other is easy again.

Showing Computational security? We have just seen that there are no provably hard trapdoor functions. As such it would be impossible to prove that any asymmetric crypto system is actually computationally secure. For example, the security of RSA depends on the fact that factoring of two primes is hard. Thus its security inherently relies on an unproven fact. However, this problem has been studied so extensively that we can be reasonably certain it is at least hard enough that an attacker will not manage to solve it in general (if they did they could win a Nobel prize instead of decrypting our data...) So perhaps we can safely *assume* that this problem is hard. But is that enough to make our crypto system secure? Clearly we have to make the right choices (large keys, etc.) but we also need to know there is no way around our system without breaking the 'hard' problem. This is where provable security comes in: we prove that breaking the cipher implies solving some supposedly hard problem.

First we need to be more precise on what 'breaking' the cipher means; what information does the attacker get and what does she need to learn to consider the cipher broken? Answering these questions requires specifying the exact setup, knowledge of the attacker and their goal. An easy to understand way to do this is by expressing security properties in the form of a game between us as the keyholder and the attacker (opponent). In the game for property IND-CPA we give the opponent the encryption (called the challenge) of one out of two texts and the opponent has to guess which one it is.

Example: IND-CPA Security Game The INDistinguishability under Chosen-Plaintext Attack (IND-CPA) security game consists of the following steps:

- The opponent picks two plaintexts.
- We randomly pick one of these two texts, encrypt it and give the result (called the *challenge*) to the opponent.
- The opponent guesses which of the two plaintexts was encrypted to get the challenge.

Of course the opponent could simply randomly guess, giving a probability of 1/2 of being correct. But perhaps the attacker can do better by examining the cipher text that we provide. How much better the attacker can do we call the *attacker advantage*: $|P(correctguess) - 1/2|$. (Note that being able to guess wrong most of the time also implies it is also possible to guess correctly most of the time, hence the use of the absolute value here.)

We say the attacker 'wins the game' if there is a distinct attacker advantage. The attacker advantage will depend on the *security parameter* which captures the 'problem size', for example the length of the key used. As the problem grows it is no longer realistic for an attacker to e.g. try decrypting with all possible keys. Formally we can thus put that a cipher is *IND-CPA secure* if the attacker advantage is negligibly small (goes to zero faster than $1/p$ for any polynomial p) for any polynomial-time attacker. Paraphrasing; the advantage of an attacker with a realistic amount of resources will go down exponentially as the key size grows.

The setup of the IND-CPA game matches a situation where the attacker has a single cipher text to examine and captures the worst case scenario for this case. If the attacker can tell that any plaintext is more likely than any other for this cipher text, this game will give her an advantage and thus allow her to win the game. Thus IND-CPA security gives us that the attacker will not be able to learn anything from a single cipher text. But what if the attacker has even more information, for example several cipher texts or even several plain-cipher text pairs? In this case she might be able to break the cipher using this extra information even when she cannot win the IND-CPA security game. To state that a cipher is suitable for such situations we need a stronger notion of security expressed by an easier to win (by the opponent/attacker) security game. There are several different security games to describe security of ciphers in different usage scenarios. The indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) security game is one such game where the attacker has access to an encryption and decryption oracle, i.e. can encrypt any plaintext and decrypt all cipher texts except, of course, the challenge itself. From the games it is clear that IND-CCA2 security implies IND-CPA security. (Any winning strategy for the IND-CPA game can also be used to win the IND-CCA2 game.)

Security Proofs As mentioned above, being able to factor large numbers implies the ability to break RSA. But is the reverse also true; is breaking RSA (i.e. winning the security game) really as hard as factoring large numbers (or another difficult problem[9])?

[9]If one is able to take the e-th root then one can break RSA. It is not known whether this is equivalent with factoring n.

A security proof of an asymmetric cryptography system shows that breaking the system (i.e. winning the security game) implies breaking some problem that is assumed to be hard. A cipher for which such a proof exists is then referred to as being 'provably secure'. Although such a proof is a very useful validation of the encryption one has to be careful in the interpretation of the term 'provably secure': it shows that breaking the cipher in a specific setting is at least as hard as some problem which is *assumed* to be hard in *general*. This by itself does not guarantee that our specific instance of the problem is not breakable (perhaps the problem was not really hard or we chose our parameters poorly (e.g. keys too small) leading to a specific instance of the problem that is easy to solve).

Even if the problem is theoretically hard you have to choose an instance that is also hard in practice; see e.g. the discussion on key choices for RSA above as an example. Factoring can become easy if you do not choose your primes carefully. Finally security proofs are difficult to make, requiring subtle probabilistic reasoning, and many mistakes have been discovered in published proofs. While there is a lot of ongoing research (e.g. there is an ongoing conference series on provable security [9]), for example to further formalize such proofs [11, 18] and make them machine checkable, provable security remains a complex issue. The same holds for translating provable to practical security [4, 19].

9.7 Summary

The goal after this chapter is that you will have a good idea of the basics of cryptography; know how symmetric and public key ciphers can be used and are able to give examples of these ciphers. It should also be clear what 'cipher X is secure' means in any given context and you should be able to check whether this is the case for a given setting and scheme.

9.7.1 Literature

Suggested reading/viewing

- Watch the first 9 minutes of the following monologue by John Oliver on the Apple-FBI 'encryption conflict': `https://youtu.be/zsjZ2r9Ygzw`. Think about who is right in this discussion (Apple or FBI) and why.
- Security Engineering Introduction [3, Ch 5].
- Handbook of applied cryptography [22, Ch 1, Sec 7.1-7.4,Sec 8.1-8.4].

9.8 Exercises (not graded: exam preparation)

1. Give four block modes for encrypting large messages.
2. Do block modes apply only to symmetric cipher or are they also relevant to asymetric ciphers? Why?
3. What are main benefits of padding with random salt when using RSA?

4. Once again consider the online music store scenario of Exercise 6 in Chapter 1. Enhance the requirements you extracted and security design that you made by considering threats and techniques introduces in this chapter. Indicate how you would apply the techniques, what trade-offs this imposes and what effect they have on both the attacks as well as on the goals of the legitimate actors in the system, and, if applicable what new goals/actors you need to introduce.

5. Decipher the following (English) text:

   Ftq Husqzqdq oubtqd ue zaf hqdk eqogdq.

6. Suppose that amongst six people each pair wants to be able to communicate securely without the others being able to eavesdrop.

   (a) How many different keys will they need in total if they use a symmetric encryption algorithm and how many if they use an asymmetric encryption algorithm?

   (b) How many keys does a person have to store secretly in both cases?

7. Four methods to encrypt multi-block messages are 'Electronic Code Book', 'Cipher Block Chaining', 'Output Feedback', and 'Cipher Feedback'. (See Figure 9.9.)

   (a) Give the corresponding decryption schemas.

   (b) CBC has limited error propagation. Explain why this is so and give the error propagation for the other schemas.

   (c) Compare the methods on the aspects secrecy, integrity and performance.

   (d) Two users notice that the under the same conditions the same message always has the same encryption and decide to randomize the encryption by prefixing message with a randomly chosen block, i.e. by encrypting R,B1,B2,...,Bn rather than B1,...,Bn.

      i. Why would you want to randomize the encryption?

      ii. For which of the methods above would this work?

      iii. Could you use the IV factor (for the methods that use one) instead? (Explain)

8. Entropy describes the amount of randomness in an unknown event/piece of data; a higher entropy indicates that more information is needed to be able to predict the outcome.

   (a) What has a higher entropy?

      i. The roll of a fair dice or an unfair dice.

      ii. A coin toss or a roll of a dice.

      iii. The state of the union address of 2008 or a pincode

   (b) The (Shannon) entropy (in bits) can be calculated with the formula $-\Sigma p(x) \cdot log_2 p(x)$ where x ranges over all possible outcomes and $p(x)$ is the probability of outcome x. Check your intuition in the previous part by calculating the entropies. (Choosing any appropriate probability distribution for the 'unfair dice'.)

(c) Show that the uniform distribution over a domain of size 2^n has an entropy of n. (This is the highest entropy that can be achieved on this domain.)

9. The one-time pad makes a message indistinguishable from messages of equal length. With letters (instead of bits) you can apply the one time pad by adding the key to the message modulo 26.

   (a) This looks a lot like the Vigenere cipher. Why does the analysis method to attack that cipher not work here?

   The encryption is secure no matter the amount of computational power the attacker has available; even trying all possible keys will not help the attacker:

   (b) Find two keys such that ciphertext 'AFIGHT' decodes to 'YESYES' and 'NONONO'.

   The one-time pad is not very convenient in use; the key is as long as the message and can only be used once.

   (c) What happens if the key of a one-time pad is reused?

   (d) Would it be possible to make an unconditionally secure cipher with a shorter or reusable key? (Remember question 8)?

Chapter 10

Analysis of Security Protocols

In this chapter, our guiding questions are:

- How do network security protocols work?
- How to analyze the correctness of a protocol?

10.1 Introduction

A protocol is a 'recipe' for communication; it describes the sequence of messages that the participants in the communication should send and earlier in the course we have seen protocols working at different layers of the network stack. Here we focus on so called *security protocols*, protocols that aim to provide secure communication over an otherwise insecure channel. Here 'secure' can mean different things. It could refer to confidentiality of the data exchanged, to proper authentication of the parties involved and their messages, to a combination thereof, etc. For Internet banking, for example, one would want both authentication and confidentiality. Obviously the requests for transfers by the client need to be authenticated but also the bank should be properly authenticated before you

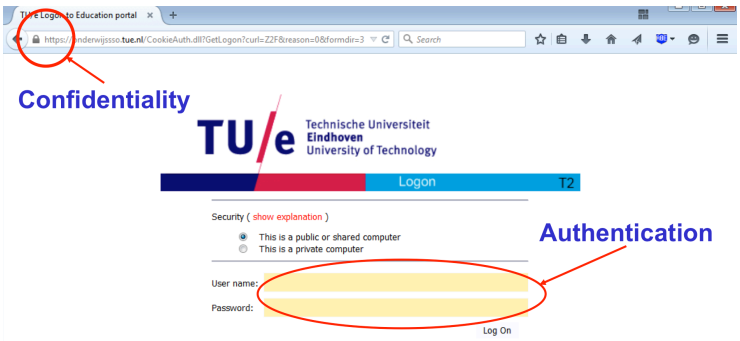


Figure 10.1: A login page over HTTPS with some security goals.

enter your login information so this information does not get stolen (for example by a phishing attack) and misused. Confidentiality is needed to protect the privacy of the client (as well as to keep the login information secret).

In earlier chapters we have seen mechanisms that can help achieve the goals of security protocols; symmetric encryption, digital signatures, etc. Engineering a good security protocol requires combining these methods in a way such that all desired security properties are achieved. In this chapter we will look at how to analyse security protocols which, as cryptography plays a large role in achieving the security properties of protocols, are also often referred to as cryptographic protocols.

10.2 Example Security Protocols

HTTPS To 'securely' browse on the web, HTTPS i.e. HTTP over a Transport Layer Security (TLS) connection is used. With HTTPS the server is authenticated (based on a certificate authority tree using certificates; see previous chapter) and confidentialilty, integrity and authenticity of the communication is provided. For the user the authenticity is a guarantee that they are indeed communicating with the correct server; the one that belongs to that webaddress (according to the certificate authority - CA). The client typically does not have certificates to prove identity; the authenticity for the webserver only means that all messages come from the same (unidentified) party. To identify the party it needs to employ other mechanisms; e.g. having the user enter a username and password (see also Section 7.2 where we addressed (user) authentication and observed the need for authentication protocols) once the connection has been established. Figure 10.1 shows an example of this. A login page is presented over HTTPS. The client can now authenticate by entering their login credentials while the TLS connection protects confidentiality of these credentials while in transit to the server.

Because the server has a (certified) public key but the client does not, in the setup of the connection we can send messages confidentially from the client to the server but not the other way around. As such the client should generate the secret (random number) on which the session key is based (any contribution from the server is to be seen as public information).[1]

SSH Another commonly used security protocol is Secure Shell (SSH) which is used to remotely log onto a server. It is a transport layer protocol that supports tunneling, in which traffic for e.g. X11 (graphical interface) connections can be forwarded over the secure SSH connection. SCP for securely copying files and SFTP, a secure alternative for FTP (File Transfer Protocol), also use SSH. SSH supports password authentication of the client but also a public key can be used; you place your public key on your account on the server. Your local SSH client can then connect to the server by using your private key instead of you entering a password. (Though you would likely want to protect your private key with a password/phrase but then against attackers with access to your machine rather than eavesdroppers on the network - i.e. a completely different attacker model.) External authentication (e.g. Kerberos, see below) is also possible.

[1] TLS is basically a new version of SSL (Secure Sockets Layer). The names are often used interchangeably.

Technically SSH is very similar to SSL/TLS and they achieve similar goals (creating a secure channel between a client and a host). SSL is mostly used for web traffic that needs to be secure, such as when transmitting valuable information (e.g. a credit card number) over the internet while SSH is often used for security executing commands remotely. SSL focuses on using certificates to authenticate the server and does not do client authentication (though this can be done over the created secure connection) while SSH has client authentication built in. The main reason for having both protocols is historical.

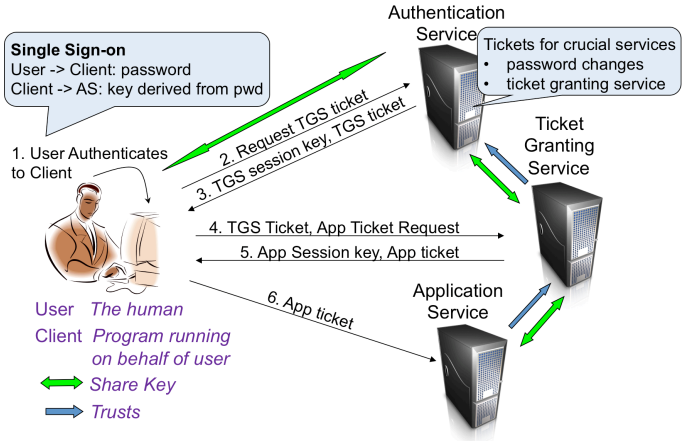


Figure 10.2: Setting of the Kerberos Authentication Protocol.

Kerberos The Kerberos network authentication protocol (illustrated by Figure 10.2) enables *single sign on* for users: after authenticating once to their local client by entering their password, users can consume services from the network without needing to authenticate (sign on) again. Commonly used on corporate networks this is convenient for the user but also eases the mangement of user credentials; i.e. user accounts and credentials are stored in a single dedicated service instead of having to be replicated to all services on the network. Other services rely on the dedicated authentication service to perform the authentication.

To give access to another service, the authentication service issues a *ticket*. This ticket enables the user to use that service. To prevent the authentication server from getting overloaded with requests it only grants tickets for the most important services (from an authentication perspective). A so called *Ticket Granting Service* (TGS) is used to grant tickets for other services. The user gets a TGS ticket for this service from the authentication service and, for the validity period of this TGS ticket, the user goes directly to the Ticket Granting Service for tickets to application services.

The actual protocol for requesting and granting a ticket works as follows: First Alice's client (A) requests a ticket from the server S for the use of server Bob (B):

$$A \rightarrow S : A, B, Texp, N_A$$

The server (S) returns a session key and an encrypted 'ticket' to give to Bob:

$$A \leftarrow S : \{K_{AB}, B, Texp, N_A\}_{K_{AS}}, \{K_{AB}, A, Texp\}_{K_{BS}}$$

Alice's client contacts service Bob:

$$A \rightarrow B : \{checksum, timestamp\}_{K_{AB}}, \{K_{AB}, A, Texp\}_{K_{BS}}$$

Alice and Bob now share a session key K_{AB} and Bob knows that this key belongs to Alice (A).

The description format used above is called an (informal) protocol narration. It specifies which roles in the protocol are supposed to take which steps. A corresponding textual description typically uses names for the letters (thus Alice instead of A, Bob for B, etc.) Thus in the Kerberos protocol the role 'user client', denoted 'A' and referred to as 'Alice' initiates the protocol by sending a message to the address of the server S. The content of this message is Alice's identity, the server she wants a ticket for B, (in the protocol narration we do not distinguish whether B is expressed as a public key, URL, ID number, IP, MAC, etc.), until when the ticket should be valid ($Texp$) and a random number N_A. She then starts listening for a response from the server.

The server, initially listening for requests, responds to Alice's message by sending information for Alice and a ticket. The information for Alice consists of a fresh session key K_{AB} for Alice and Bob to use. As this key needs to remain secret it is encrypted (notation: $\{\ldots\}_K$) with the key K_{AS} that the server and Alice already share. Alice can decrypt the message and check the random number N_A and she learns K_{AB}. Of course this key will look like any random string to her so B is included in the message to confirm to Alice that this is for talking to Bob. (Whether this is sufficient to prevent Alice from accepting an incorrect key K_{AC} is not directly obvious. Below we look at analysis of protocols in more detail to be able to answer this type of question.)

Alice cannot decrypt the ticket $\{K_{AB}, A, Texp\}_{K_{BS}}$ as she does not have the key K_{BS}; only Bob and the server have this key. Yet she can forward the ticket to Bob along with an extra message that $\{checksum, timestamp\}_{K_{AB}}$ which confirms that she also knows K_{AB}.

Note that once Alice and Bob share a key Bob can take the role of server in a new run of the protocol. This occurs, for example, if Bob is the Ticket Granting Service; Alice will contact server Bob to get a ticket for some other service Charlie.

Kerberos can be used, not only within a single 'realm' (the services under auspices of the authority which controls the authentication service) but also across realms. (See Figure 10.3) A client in realm 1 wanting to use a service in realm 2, will get a ticket for the authentication service of realm 2 from its own authentication service. With that ticket it can then ask for a ticket of the realm 2 service (which in turn may be the Ticket Granting Service for realm 2).

So what is the role of the random number N_A? Do we need to include B in A's information and A in the ticket for B? Should we include more, such as also B in the ticket for B? To be able to answer these questions we need to consider the security goals of the protocol and how we can analyse the protocol. This is the topic of the following sections.

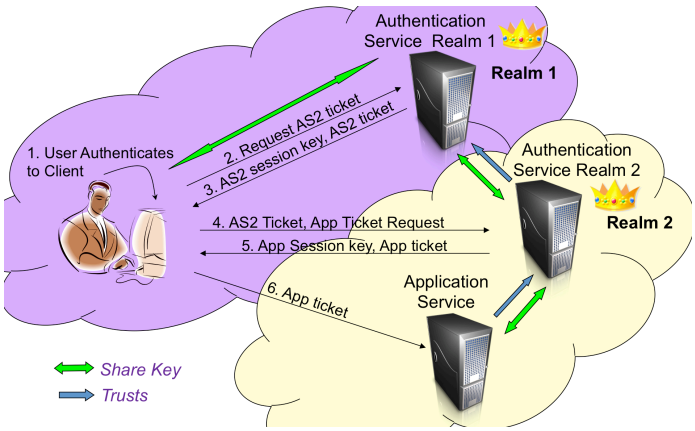


Figure 10.3: Cross domain authentication with Kerberos.

10.3 What makes a security protocol tick: Protocol design

Above we have seen some examples of commonly used protocols. But what makes a protocol work; how are the security goals achieved? Let's try to construct a very simple protocol for remote authentication of a user (Alice) to a server (Bob). Recall from Section 7.2 that authentication of end users can be done using different factors; what you know, are or have. In remote e-authentication you use tokens. But how do you prove control over your token. This will typically involve a secret related to the token. If the token is your password (a 'what you know' token) this is the secret involved in proving control. To remotely prove that you have something (e.g. a credit card) you will likely use a secret of that object (e.g. the numbers printed on it). If using a biometric ('what you are') token your biometric device would recognize you and likely use a secret, such as a private key, to confirm your identity to a relying party. Thus an authentication protocol is typically designed to prove control by the claimant over some form of *secret* to a remote verifier.

Let us assume Alice knows a secret (such as a password) which is also stored on the server and when Alice wants to authenticate she sends the secret. If the attacker is not able[2] to see messages between Alice and the device and cannot guess the secret (see e.g. the discussion on passwords in Section 7.2) then this solution works. On a computer network, however, we definitely need to consider attackers that are able to eavesdrop on the communication. With such a slightly stronger attacker model this approach is clearly not secure - after the first time Alice authenticates the attacker also has the secret. Simply encrypting the secret will not help; the attacker does not learn the secret but can simply send the same message again (a so called *replay attack*) to authenticate. The verifier needs to know that a claimant in control of the token is involved now and not at some point in the past; *freshness* of the proof of control is needed.

A challenge-response mechanism can be used instead to prove that Alice has the secret without revealing it. The server sends a challenge (e.g. a ran-

[2]Remember security depends on the attacker model.

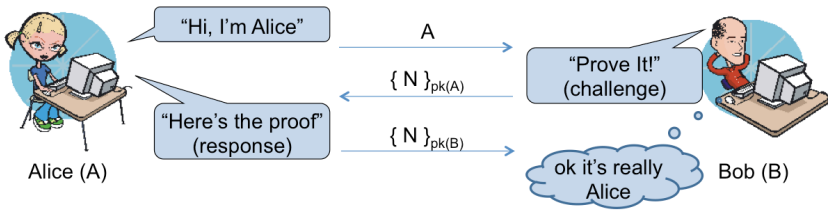


Figure 10.4: Basic authentication protocol, first attempt.

dom number $B \rightarrow A$: *challenge*) and Alice's response should be one that only Alice can make and only after Alice has received the challenge ($A \rightarrow B : response(challenge, secret)$). In this way upon receiving the response, the server can be sure that Alice got the challenge and that Alice's response is fresh and not a response that Alice sent out in some earlier session. To send the challenge the server needs to know that Alice wants to authenticate so she will first have to alert the server ($A \rightarrow B : A$).

Of course the server does not need to have the secret of Alice, only a way of making a challenge that only someone with the secret (Alice) can answer and of checking that the response is correct. Thus for our protocol let us assume Alice's secret is a secret key $sk(A)$ and that Bob knows Alice's public key $pk(A)$. Our first attempt at the protocol could then be: Bob's challenge $\{N\}_{pk(A)}$ is new random number N (called a nonce) encrypted with Alice's public key. Alice proves that she knows $sk(A)$ by decrypting the challenge and sending back the random number.

Here Alice is only trying to authenticate. However, usually the goal will be to also exchange of some information such as a secret number that can be used to create a session key. If we adapt the protocol as shown in Figure 10.4 by encrypting the final message with the public key of Bob ($A \rightarrow B : \{N\}_{pk(B)}$) we might expect to also achieve *secrecy* of the nonce N.

Is the protocol correct? After running the protocol is server Bob really 'talking to Alice' and, for the adapted protocol, is N a secret known only by Alice and Bob? One thing that the server does not know is how far away Alice is. When remotely accessing some service on the network it likely does not matter how far away the user is. When running a protocol between e.g. an OVchip card and a metro access gate one would like to be sure that the user (card) is actually at the gate. If a response comes within a millisecond the challenge and response travelled at most (cannot exceed the speed of light) 0,001s * 300.000 km/s = 300km thus the responder is at most 150km away. To actually get useful distance bounds one needs to use more advanced *distance bounding protocols* which are outside the scope of this course. (See the master course Physical Aspects of Digital Security (2IC35) if you want to learn about distance bounding protocols.) In the protocol analysis treated here we will assume the location is irrelevant.

For authentication over the network Bob is 'remotely' talking to Alice. As there is a network in between both Alice and Bob are (and are supposed to be) talking to some routers rather than (directly) to each other. We will thus define authentication to have correctly been completed if when Bob thinks he has successfully completed the protocol with Alice then Alice did indeed run the

Suppose that Alice uses an online banking service Bob and a subscription based joke-a-day service Mallory. Both services use our protocol for authentication. When Alice authenticates to Mallory to get her joke of the day,

$$A \rightarrow M : A$$

Mallory misuses this fact and will try to pretend to be Alice to Bob

$$M(A) \rightarrow B : A$$

Bob sends his challenge back to 'Mallory pretending to be Alice' (or directly to Alice depending on how the routing works; it does not matter as Mallory can intercept all messages).

$$B \rightarrow M(A) : \{N\}_{pk(A)}$$

Mallory cannot decrypt the message … but she can forward it to Alice …

$$M \rightarrow A : \{N\}_{pk(A)}$$

Alice is waiting for a challenge from Mallory and will dutifully respond.

$$A \rightarrow M : \{N\}_{pk(M)}$$

Now this message Mallory can decrypt. She gets N and answers Bob's challenge.

$$M \rightarrow B : \{N\}_{pk(B)}$$

Now Mallory can empty Alice's bank account and 'the joke is at the expense of Alice...'

Figure 10.5: A 'Mallory in the Middle' attack on our protocol.

protocol and with the intention of authenticating to Bob. Secrecy of information, of the nonce in this case, is easier to define; no one other than the intended parties (Alice and Bob in this case) should learn the information.

Definition 10.3.0.1 *A protocol is said to achieve authentication of party A to party B if whenever B completes the protocol, apparently with A, A started the protocol, apparently with B.*

With the definition in place we can return to our original question; is the protocol correct, does it achieve authentication of Alice? Of course this question is not yet complete; we have defined the security goals but we also need to give the attacker model.

As the network is a dangerous place (see Chapter 8) we play it safe in our attacker model and assume the worst case scenario in which the attacker, we will call her Mallory, has full control over the network; she can intercept all messages, block or change them, and create new messages. The attacker may also have a valid identity (or even several) on the network with which she can partake in protocol runs. However, we also need to assume some limits on Mallory's abilities. If she could also extract the nonce from the challenge message $\{N\}_{pk(A)}$ without knowing Alice's private key she can easily break the authentication; she can send message A to Bob, intercept Bobs response, extract the nonce N and return it to Bob. However, this is a problem with the

cryptography, and not with the protocol! We want to separate these concerns; analyze the protocol without complicating matters too much with the details of how the cryptography works. Thus in the protocol analysis we typically assume that the cryptography simply works (the so called *perfect cryptography* assumption).

Therefore our complete question becomes: *Does the protocol achieve authentication of Alice when the attacker has full control over the network but cannot 'break the cryptography'?* It may seem to work correctly at first glance; only Alice can extract the nonce N from the message and, as it is fresh, when Bob receives the nonce back he knows that Alice did this extraction after he sent it. Thus Alice is alive and well and is answering the challenge. Yet what Bob does not know is whether Alice is actually trying to authenticate to Bob. Perhaps Mallory somehow tricked Alice into answering the challenge. The attack in Figure 10.5 shows how Mallory might go about this.

The Man (Mallory) in the Middle attack (Figure 10.5) on our protocol works as follows: Alice wants to authenticate to Mallory and when Alice sends her authentication request to Mallory, Mallory sends one to Bob pretending to be Alice in the protocol run with Bob. Bob will send a challenge to check whether Mallory is actually Alice. Mallory cannot answer this challenge herself but can forward it to Alice (pretending it came from Mallory herself). As Alice is waiting for a challenge to come from Mallory, she is trying to authenticate to Mallory after all, she will answer this challenge, decrypting the nonce and returning it to Mallory encrypted with Mallory's public key. In this way Mallory obtains the nonce and can now answer the challenge from Bob by encrypting the nonce with his public key.

Even in this simple example we managed to get it wrong. It seems to be trickier than it looks. This is a general problem; even though protocols may seem simple, it is easy to make mistakes.

> "*Security protocols are three line programs that people still manage to get wrong.*"
> – Roger Needham –

We could try to fix the flaw but how can we be sure that our fix works and there are no further mistakes. Formal analysis supported with automated ways of checking models or proofs, can help to find mistakes. Formal analysis forces you to be very precise in your assumptions and descriptions and automatically checks (or even generates) the conclusions you can draw from them. It is important to remember that, as we also see with 'provable secure cryptography', a formal proof is only as good as its input. If we make mistakes in modeling the protocol and its context, or incorrect assumptions about how the protocol is used, the attacker's initial knowledge and capabilities, etc. a formally verified protocol may still be vulnerable. Thus formal analysis is very useful and needs to be done, but we should not blindly rely on its results.

Formal analysis To do formal analysis we have to be very precise on what our security model is. We already considered security goals such as authentication and secrecy. As attacker model we use the *Dolev-Yao* model (see Figure 10.6). This model captures the idea we stated above that the attacker has full control over the network but cannot break the cryptography.

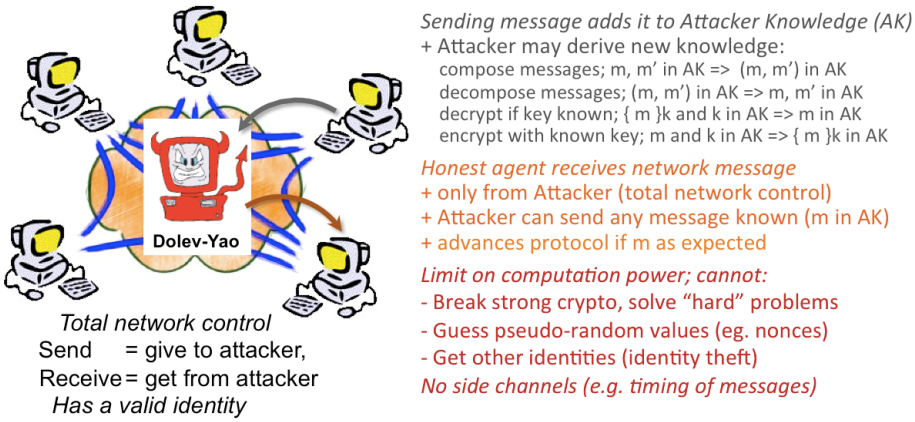


Figure 10.6: The Dolev-Yao attacker model.

The Dolev-Yao model is very suitable for automated formal analysis. As long as the number of (potential) participants is finite, verification of properties like authentication is possible (decidable) even when considering an arbitrary number of possible messages. An alternative attacker model is the computational attacker where an attacker is an arbitrary probabilistic Turing machine. With such an attacker model we can also consider the cryptographic primitives being used (is encryption schema x combined with random number generator y suitable for protocol z?) but it is much harder to reason about. The analysis could look like that in Section 9.6 except with a more complicated security game which captures both the security goals and the protocol. Here we will stick with the Dolev-Yao model, see Figure 10.6.

Crucial to analysing a protocol is the attacker knowledge. As the attacker has full control over the network, sending any message will add that message to the knowledge of the attacker. The attacker can combine this message with the knowledge she already has; for example by decrypting a message if she knows the key.

Note that the analysis of a security protocol can be quite subtle. Consider for example Figure 10.7. The sequence in that figure looks very similar to the man in the middle attack that we see in the Figure 10.5 but is actually quite different. Here Mallory is actually a good friend faithfully passing along messages between Alice and Bob. (Recall that in our attacker model sending is giving the message to Mallory and we can only receive them from Mallory; Mallory is the network.) Alice is correctly authenticated to Bob, whom she is actually trying to talk to and Mallory does not obtain the nonce.

Then how can we protect against the man in the middle attack we found? The problem comes from the fact that Alice does not know that the challenge came from Bob. We can solve this by simply telling her; including the identity of the challenger in the challenge. Now Mallory's attack will no longer work; when she forwards the challenge to Alice, Alice will see that it is a challenge from Bob and not the challenge from Mallory that she was expecting - she will thus not answer it. Note that Alice cannot be sure that the challenge came from Bob as anyone can construct such a challenge; however, Bob when getting a response to his challenge, can know for sure that Alice intended this to be for

In the following flow Mallory is not an attacker but a faithful network

$$
\begin{aligned}
A &\to M(B) : A \\
M(A) &\to B : A \\
B &\to M(A) : \{N\}_{pk(A)} \\
M(B) &\to A : \{N\}_{pk(A)} \\
A &\to M(B) : \{N\}_{pk(B)} \\
M(A) &\to B : \{N\}_{pk(B)}
\end{aligned}
$$

Authentication (of Alice to Bob) is achieved and as Mallory learns nothing about the nonce N, secrecy of the nonce N is also satisfied.

Figure 10.7: Mallory being a faithful network.

him.

Patched protocol and attempted attack. The message of Mallory does not match the expectation of Alice so she does not advance the protocol; Mallory does not learn N.

$$
\begin{aligned}
A &\to B : A & \qquad A &\to M : A \\
B &\to A : \{N, B\}_{pk(A)} & M(A) &\to B : A \\
A &\to B : \{N\}_{pk(B)} & B &\to M(A) : \{N, B\}_{pk(A)} \\
& & M(A) &\to A : \{N, B\}_{pk(A)} \\
& & A &\not\to M : (\text{was expecting} \{?, M\}_{pk(A)})
\end{aligned}
$$

Figure 10.8: Patched protocol protects against the attack.

The adjustment makes the attack above impossible. But are there any other attacks that are still possible? Ideally we would create a formal proof but this is outside of the scope of this course; the master course *Verification of Security Protocols* (2IF02) addresses this topic. Yet we can reason using the Dolev-Yao model. An informal argument for correct authentication (of Alice to Bob) should at least argue the following points:

1. A secret of Alice is used; a challenge that only Alice can answer ensures that the attacker cannot complete the authentication without involving Alice.
2. Freshness of Alice's response; the secret has to be used in this session and not be e.g. replayed from an earlier session.
3. Alice's response is meant to authenticate her to Bob; when Alice is not trying to authenticate to Bob it should not be possible to trick her into answering the challenge for the attacker.

We addressed point 3 above by adding Bob's name in the challenge that is sent to Alice. But is this always sufficient to prevent Alice from being confused?

Let us consider another example protocol: the Otway-Rees Protocol for session key distribution using a trusted server (see Figure 10.9). This protocol illustrates a different method of authentication; via a trusted third party and also allows us to illustrate another possible attack against protocols.

Otway-Rees and type flaw attacks The Otway-Rees protocol uses a trusted server to create a short term session key (Kab) for communication between Alice and Bob. Alice and Bob already share (long term) keys (Kas and Kbs respectively) with the trusted server. The server is trusted in that we assume it will behave correctly; generate good fresh random keys, not leak information to the attacker (other than what is leaked by correctly following the protocol) nor misuse the information it has. The server, for example, will know the key used by Alice and Bob to communicate but this is not considered to be a problem. Note that the protocol uses only symmetric cryptography. The key that people share with the server is used to authenticate them rather than a public-private key pair.

The Otway-Rees protocol:

1. $A \rightarrow B : Ms, A, B, \{N_a, Ms, A, B\}_{Kas}$
2. $B \rightarrow S : Ms, A, B, \{N_a, Ms, A, B\}_{Kas}, \{N_b, Ms, A, B\}_{Kbs}$
3. $S \rightarrow B : Ms, \{N_a, Kab\}_{Kas}, \{N_b, Kab\}_{Kbs}$
4. $B \rightarrow A : Ms, \{N_a, Kab\}_{Kas}$

The attack by Mallory:

1. $A \rightarrow M(B) : Ms, A, B, \{N_a, Ms, A, B\}_{Kas}$
4. $M(B) \rightarrow A : Ms, \{N_a, Ms, A, B\}_{Kas}$

A expects to get $Ms, \{N_a, Kab\}_{Kas}$ in step 4 so she may misinterpret (Ms, A, B) as being the key Kab. Mallory also knows this 'key'; she knows Ms, A, and B so can make (Ms, A, B).

Figure 10.9: A type flaw attack on the Otway-Rees protocol

A so called *type flaw attack* is possible against Otway-Rees (also presented in Figure 10.9). The attacker sends a message to Alice which is of a different type than she is expecting. Depending on the implementation, however, Alice may not be able to tell that this is the case. As part of step 4, Alice is expecting to receive her nonce and a key encrypted with the key that she shares with the server. An attacker will not be able to create such a message. However, there are other messages that are encrypted with this key that the attacker could use: the first message of the same run of that protocol for example. If the attacker sends back Ms and $\{N_a, Ms, A, B\}_{Kas}$, both of which she got as part of message 1 then it will contain the right nonce (N_a). The remainder of the message (Ms, A, B) could then by mistake be accepted as being Kab.

To protect against type flaw attacks we can tag messages with their type, e.g. using $(K_{ab}, \text{`}key\text{'})$ rather than K_{ab}. Of course we still need to ensure that this tagging is sufficient to prevent confusion. The general lesson to take away is

that care needs to be taken to ensure that messages cannot be misused elsewhere in the protocol.

Let us return to authentication but now both ways, i.e. mutual authentication. The Needham-Schroeder protocol is a well known protocol that has been used for this purpose.

The Needham-Shroeder protocol (1978) for mutual authentication:

$$1.\ A \to B : \{A, N_a\}_{pk(B)}$$
$$2.\ B \to A : \{N_a, N_b\}_{pk(A)}$$
$$3.\ A \to B : \{N_b\}_{pk(B)}$$

Low discovered a vulnerability in 1995(!) through an analysis using formal methods. The vulnerability is very similar to the one in our one-way authentication protocol. The patched protocol (called NSL) replaces the second message by:

$$2.\ B \to A : \{N_a, N_b, B\}_{pk(A)}$$

Figure 10.10: The Needham-Schroeder and Needham-Shroeder-Low protocols

Needham-Schroeder-Low The Needham-Schroeder protocol, see Figure 10.10, aims to achieve mutual authentication between Alice and Bob. It basically runs the authentication protocol we designed before in both directions. It also aims to maintain secrecy of the nonces used so they can be used to create a session key.

It is vulnerable to an attack very similar to the one we saw before; Alice can be tricked into answering a challenge from Bob even though she is not trying to authenticate to Bob but rather to Mallory. The patch is also the same; add the identity B to the challenge from B. This patch solves this specific attack. On NSL there is still a potential type flaw attack, however the type confusions that need to happen seem unlikely to occur in practice.

A type-flaw attack on NSL involving three sessions (I,II,III) where Alice wants to authenticate to Bob (I), Mallory misuses this to fake being Alice to Bob in (II) and Mallory pretends to want to authenticate to Alice (III) to help achieve this:

$$\text{I.1.}\ A \to M(B) : \{A, N_a\}_{pk(B)}$$
$$\text{II.1.}\ M(A) \to B : \{A, M\}_{pk(B)}$$
$$\text{II.2.}\ B \to M(A) : \{M, N_b, B\}_{pk(A)}$$
$$\text{III.1.}\ M \to A : \{M, (N_b, B)\}_{pk(B)}$$
$$\text{III.2.}\ A \to M : \{(N_b, B), N'_a, A\}_{pk(M)}$$
$$\text{II.3.}\ M(A) \to B : \{N_b\}_{pk(B)}$$

Figure 10.11: A type-flaw attack on the Needham-Shroeder-Low protocol

The type flaw attack, shown in Figure 10.11, involves two type confusions. The first is that Bob misinterprets M in message II.1 as a nonce. He responds with message $\{M, N_b, B\}_{pk(A)}$ to prove he knows M, which he thinks is a nonce from A, and adds his own nonce N_b as a challenge to A. Mallory cannot decrypt this message, however, she can start a new session (III) with Alice and send this message as the first message of this new session. Alice expects new sessions from Mallory to start with a message of the form $\{M, ?\}$ where ? is the challenge (nonce) from M. She gets $\{M, N_b, B\}$. Thus, in a second type confusion, she may interpret all of (N_b, B) to be the nonce from Mallory. Responding to this challenge she sends $\{(N_b, B), N'_a, A\}_{pk(M)}$ to prove she knows Mallory's challenge (N_b, B) and give her own challenge to Mallory (N'_a). Now Mallory can decrypt this message and extract everything including N_b. Knowing N_b she can finally answer the challenge from Bob. Mallory has successfully pretended to Alice and to Bob that she knows the 'nonces' (M and N_b) that will be used to build the key for this session.

Privacy Our authentication protocol design (e.g. Figure 10.8) introduces some privacy concerns. First of all, Alice's identity is sent in the clear. The protocol aims to authenticate Alice only to Bob, not let everybody listening on the network know that Alice wants to talk to Bob. For our protocol we can solve this by encrypting Alice's identity with Bob's public key. Yet even then there is a potential privacy issue: an attacker could determine whether Bob is willing to talk to Alice. Anyone, including the attacker, can send the message $\{A\}_{pk(B)}$. Even though the attacker cannot answer the challenge from Bob, the very fact that Bob sends a challenge is sufficient.

Bob is only willing to talk to some parties and only those parties should be able to learn when he is willing to talk to them.

$$1.A \to B : \text{`}hello', \{\text{`}hello', N_a, pk(A)\}_{pk(B)}$$
$$2.B \to A : \text{`}ack', \{\text{`}ack', N_a, N_b, pk(B)\}_{pk(A)}$$
$$2'.B \to A : \text{`}ack', \{N\}_K$$

If Bob is not willing to create a session with Alice he sends a decoy response ($2'$). Here K is any random key.

Figure 10.12: Private authentication protocol (Abadi 2002)

A possible solution to hide Bob's choice is to always send a reply in such a way that only Alice will be able to tell the difference. Abadi's private authentication protocol, see Figure 10.12, tries to achieve this. For this protocol to work the decoy message should look real (be indistinguishable from a real message for the attacker). Also the way we handle the decoy message should look real. For example, if the decoy is easier to make than the real response and we respond too quickly then the attacker may deduce the message is a decoy. (This is an example of the so called *side-channel* information.)

Here we have mainly considered preventing Mallory on her own (so without Alice being active) being able to tell whether Bob will talk to Alice. If Alice herself connects to Bob, Mallory may be able to tell Bob's choice by the fact that

a.1. $A \rightarrow B : \{A, N\}_{pk(B)}$ $\quad$ b.1. $A \rightarrow B : \{A, s\}_{pk(B)}$
a.2. $B \rightarrow A : N$

Alice and Bob use a protocol for authenticating Bob (a) and Alice wants to send secret messages to Bob (b). When Alice sends her secret Mallory can obtain it by using the other protocol:

b.I.1. $A \rightarrow M(B) : \{A, s\}_{pk(B)}$
a.II.1. $M(A) \rightarrow B : \{A, s\}_{pk(B)}$
a.II.2. $B \rightarrow M(A) : s$

Figure 10.13: An attack across multiple protocols.

Alice and Bob continue sending each other messages after the authentication protocol completes. Preventing Bob's choice from leaking in such situation requires more advanced anonymization techniques.

Use of multiple protocols As a final aspect let us consider the use of multiple protocols. Two protocols that are by themselves correct may became vulnerable/flawed when combined. A basic challenge response protocol using the public key of Bob, combined with a protocol that relies on secrecy of messages encrypted with this public key are obviously flawed when combined. (See for example Figure 10.13). To address this we should analyse all protocols that may be used together. Using different keys for different protocols would be one way to avoid confusion of at least the encrypted parts of the messages between protocols. The obvious drawback is that users need to have multiple keys.

10.4 Summary

It is hard to design a good protocol. Attacks such as man in the middle or type flaw can break security goals of a protocol such as authentication and secrecy. Very subtle flaws can have big consequences. This also makes the analysis of protocols a difficult exercise. Analysis by formal methods is needed to provide some degree of rigor.

10.4.1 Literature

Suggested reading The Spore repository contains many examples of protocols and attacks against them. You do not need to know these protocols by heart but should be able to understand a given protocol specification and analyse it to find attacks.

- The Security Protocol Open Repository (Spore) [16].
- Security Protocols and Their Properties [2].

10.5 Exercises (not graded: exam preparation)

1. Describe the Dolev-Yao attacker model; what can the attacker do in this model and what not?

2. Consider the following protocol: Alice send an encrypted message to Bob, along with her public key. Bob responds by sending a fresh (symmetric) key to Alice encrypted with her public key. Finally Alice sends Bob the key used to encrypt the message encrypted with Bob's fresh key.

   (a) Write down the protocol in protocol narration form.

   (b) Does the protocol achieve secrecy of the message?

3. Translate the following protocol narration to English text:
   $A \to S : \{N_A, K\}_{K_{AS}}, A, B$
   $S \to B : \{A, N_A, K\}_{K_{BS}}$
   $B \to A : \{N_A\}_K$

4. Consider the man (Mallory) of the middle attack in Figure 10.5. Write down the attacker knowledge in each step. Use a canonical form, i.e. write only terms that cannot be decomposed any further by Mallory.

5. Consider again the online music store of the previous chapters. Review your requirements analysis, adding security protocol considerations; threats and countermeasures where appropriate.

6. Alice has a public key known to everyone. Bob knows a secret s_B (eg a password) that Alice will recognize as being Bob's when she sees it (eg because she has the hash of the password).

   (a) Alice wants to send a secret message m to Bob. Write a protocol to get the message to Bob securely and argue why your protocol achieves this.

   (b) What if we also want to authenticate the message, let Bob know for sure the message came from Alice; adapt your protocol if needed and argue why the message is (now) also authenticated.

7. Consider again the protocol
   $A \to S : \{N_A, K\}_{K_{AS}}, A, B$
   $S \to B : \{A, N_A, K\}_{K_{BS}}$
   $B \to A : \{N_A\}_K$
   Assuming that server S is trusted, does the protocol achieve:

   (a) authentication of Alice to Bob,

   (b) authentication of Bob to Alice,

   (c) secrecy of N_A ?

   How could you fix the protocol for the properties above not yet achieved?

8. A flawed mutual authentication protocol:

   (a) A->B: A, K

(b) B->A: {B, K, Nb}pk(A)

(c) A->B: {Nb}K

where A and B are agent identities, K is a fresh symmetric key and Nb is a nonce, pk(X) is the public key of agent X, and {M}pk(X) is message M encrypted with the public key of X using a public-key algorithm (as in message 2).

The protocol above aims at mutual authentication of agents A and B. Authentication of B to A is not accomplished, since A is not really presenting a challenge to B.

(a) Provide an attack that allows the intruder to impersonate agent B and fool A into thinking she's communicating with B

(b) Provide a fix for the previous flaw, and explain why you think it avoids the mentioned vulnerability.

(c) Which secrets, if any, are shared between A and B after the original protocol and after the fixed protocol?

(d) Is authentication of A to B accomplished? (Explain your answer.)

(e) Is the "B" really necessary in message 2. B->A: {"B",K,Nb}pk(A)? (Explain your answer.)

Bibliography

[1] https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/.

[2] M. Abadi. Security protocols and their properties. In *NATO Science Series: Volume for the 20th International Summer School on Foundations of Secure Computation*, pages 39–60, Marktoberdorf, Germany, 1999. http://www.cse.ucsc.edu/~abadi/allpapers.html#marktoberdorftoo.

[3] Ross Anderson. *Security Engineering*. Wiley Computer Publishing, 1991. www.cl.cam.ac.uk/~rja14/book.html.

[4] Mihir Bellare. Practice-oriented provable-security. In *Information Security*, pages 221–231. Springer, 1998.

[5] S. Bellovin. Defending against sequence number attacks, 1996. IERF RFC 1948 http://www.ietf.org/rfc/rfc1948.txt.

[6] K. Böhm, S. Etalle, J. I. den Hartog, C. Hütter, S. Trabelsi, D. Trivellato, and N. Zannone. A flexible architecture for privacy-aware trust management. *Journal of Theoretical and Applied Electronic Commerce Research*, 5(2):77–96, August 2010.

[7] William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, and Emad A. Nabbus. Electronic authentication guideline. NIST Special Publication 800-63-2, https://doi.org/10.6028/NIST.SP.800-63-2.

[8] F. Charpentier. CCWAPSS 1.1 whitepaper. http://www.xmco.fr/whitepapers/ccwapss_1.1.pdf.

[9] Sherman SM Chow, Joseph K Liu, Lucas CK Hui, and Siu Ming Yiu. *Provable Security: 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782. Springer, 2014.

[10] OASIS committee on XACML. A brief introduction to xacml. http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html.

[11] R. J. Corin and J. I. den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In M. Bugliesi, B. Preneel, and V. Sassone, editors, *ICALP 2006 track C, Venice, Italy*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263, Berlin, July 2006. Springer-Verlag.

[12] E. Costante, J.I. den Hartog, M Petkovic, S. Etalle, and M. Pechenizkiy. Hunting the unknown - white-box database leakage detection. In *Proceedings of 28th IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSEC2014)*, volume 8566 of *LNCS*, pages 243–259, 2014.

[13] Marcin Czenko, Sandro Etalle, Dongyi Li, and William H Winsborough. An introduction to the role based trust management framework rt. In *Foundations of security analysis and design IV*, pages 246–281. Springer, 2007. `doc.utwente.nl/64136/1/IntroRTFinal.pdf`.

[14] W. Diffie and M.E.Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer magazine*, 1997. `http://www-ee.stanford.edu/~hellman/publications/27.pdf`.

[15] Golnaz Elahi and Eric S. K. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *ER*, pages 375–390, 2007.

[16] Laboratoire Spécification et Vérficication. Security protocol open repository (spore). `http://www.lsv.ens-cachan.fr/Software/spore/table.html`.

[17] Behrouz A. Forouzan. *Data Communications and Networking*. McGraw-Hill Higher Education, 4th edition edition, 2007.

[18] J.I. den Hartog. Towards mechanized correctness proofs for cryptographic algorithms: Axiomatization of a probabilistic hoare style logic. *Science of Computer Programming*, 74(1-2):52–63, 2008.

[19] Neal Koblitz and Alfred J Menezes. Another look at" provable security". *Journal of Cryptology*, 20(1):3–37, 2007.

[20] James F. Kurose and Keith W. Ross. *Computer Networking. A Top-Down Approach*. Pearson, 6th edition edition, 2012.

[21] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 114–, Washington, DC, USA, 2002. IEEE Computer Society.

[22] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 5th edition, 2001. `http://cacr.uwaterloo.ca/hac/`.

[23] OASIS Standard. extensible access control markup language (xacml) version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[24] Brett Stone-gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover.

[25] F. Turkmen, J.I. den Hartog, S. Ranise, and N. Zannone. Analysis of xacml policies with smt. In *4th Conference on Principles of Security and Trust (POST 2015)*, 2015. to appear.

[26] F. Turkmen, J.I. den Hartog, and N. Zannone. Poster: Analyzing access control policies with smt. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS 2014)*. ACM, 2014.