Network systems

Notes on the lectures

Week 7

Congestion control

The queue in a network is finite, meaning packets will be dropped if the queue is full. Hosts and routers then may need to retransmit them. Sending packets into a network at too high a rate causes congestion, which has a few consequences:

- Long queues in routers → long delays;
- Overflow of queues in routers → packet loss;
- Lost packets need to be retransmitted;
- If a packet is dropped at a router, the bandwidth until that router is wasted;

A long packet delay may cause a timeout in protocols like TCP, making a retransmission unnecessary. Infinite buffers are not a solution. Without feedback about congestion, the network may suffer congestion collapse.

Congestion control approaches exist to combat congestion. Such an algorithm needs to know a few things, and learn them automatically:

- The optimal sending data rate;
- How to adapt to changes;
- How to share bandwidth in a fair way;

Congestion control already exists in TCP. TCP utilizes a host-centric, feedback-based, window-based congestion control approach. Hosts try to detect congestion by observing what happens in the network, a dropped packet is treated as a sign of network congestion. With these observations, the hosts adjust the congestion window. The hosts never send more data than allowed by the congestion and receiver window.

Additive Increase, Multiplicative Decrease (AIMD) increases the congestion window by 1 MSS (Maximum Segment Size) every RTT. If a packet loss is detected, the congestion window is halved.

A good way to find the available network capacity quickly is by utilizing a slow start. Every time a packet is lost, the slow start threshold is set to half the congestion window, and the congestion window gets set to 0. If the congestion window is smaller than the slow start threshold, the congestion window is doubled every RTT. If the congestion window is larger than the slow start threshold, the congestion window is increased by 1 every RTT.

Fast retransmit, fast recovery is another method TCP uses. A packet is retransmitted after seeing three duplicate ACKs, instead of after a timeout. Fast recovery means that the slow start is skipped after a fast retransmit.

Resource allocation & Quality of Service

TCP needs packet loss to discover congestion. This is usually caused by buffer overflow. Overflows can be avoided by using the RTT to measure the average queue length, randomly dropping packets before the buffer actually overflows, and by explicitly notifying about congestion.

Random Early Detection (RED) lets the router observe the average queue length to detect congestion. To notify the end-hosts, the router randomly drops packets already before the overflow occurs. RED is an example of Active Queue Management (AQM), a technique which lets sources reduce their sending rate already, before the buffers really overflow.

The idea of Explicit Congestion Notification (ECN) is to let routers tell end-hosts explicitly that there is a congestion, rather than implying it by dropping packets. Possible approaches for ECN include literal notifications being sent back to the source, modifying the packets to the destination and letting it communicate with the source, or specifying special packets that sources can send to request the level of congestion from the router. The Internet uses the second approach.

ECN in IP uses two unused bits in the IP header. 00 (the default) means the transport layer does not support ECN, 01 and 10 indicate that it is ECN-capable, and 11 indicates that the layer is ECN-fcapable, *and* that congestion has been experienced. An uncongested router does not change these bits. A congested router drops packets if the transport layer is not ECN-capable, or sets the bits to 11 if the transport layer is ECN-capable.

If the destination receives the message that some element in the network is congested, it needs to report this to the source. In TCP this is done by using an unused bit as the ECN-echo bit (ECE).

Some applications require assurance that their data is delivered on time. Examples of such real-time applications include audio and video streaming where any delay or drop is unacceptable.

IP, and UDP, transport packets with best effort. There are no guarantees on delivery or delay. TCP uses retransmissions to provide delivery guarantees, but there is a possibility of a very long delay if packets are often dropped. This makes TCP unsuitable for real-time applications. That’s why real-time applications use UDP, and pay attention to jitter and packet loss.

There are two strategies to ensure that the network meets the applications’ requirements. The first is to simply make sure it does, over provisioning the network. This is not very efficient. The second strategy is to treat packets from different applications differently, or biased. This is called QoS (Quality of Service) differentiation. The implications are that an application that is tolerant for delays may have lower priority than an intolerant application, like VoIP.

QoS-solutions need to consider scarce resources, and make decisions about those scarce resources. Such decisions include the order of packets, and when to order what packets in the queue.

There are a lot of possibilities to decide the order of packets in the queue, some more fair than others. In FIFO (First in, first out), packets are simply sent out in the order they enter the queue. Priority queuing uses two queues; packets with low priority are sent whenever the high-priority queue is empty. Round-robin scheduling classifies packets on their priority in different queues, and a single packet from each class is served in rounds. Round-robin is only fair when all the packets have equal sizes. Finally, an imaginary solution is bit-by-bit round-robin, which transmits one bit from each class per round. Although completely fair, this is not possible in practice.

Fair queuing tries to approximate bit-by-bit round-robin as close as possible. Packets are ordered such, that they leave the queue in the same order as they would in bit-by-bit round-robin, but they are never interrupted. Weighted fair queuing is an extension of fair queuing, using a weighing factor. It makes it possible to divide the bandwidth in unequal parts.

The token bucket specification specifies a **model** that guarantees a certain average bandwidth that a certain flow uses. As it's quite difficult to explain in text, simply take a look at slides 26 up to 27 of the Wednesday lecture of week 7, page 7.

In the Internet, three QoS approaches can be used. A naive approach is to simply provide more than enough bandwidth, and hope for the best. This does not work; it's expensive and has an awful yield.

Integrated services (IntServ) reserve capacity on every link for individual flows that need it. Admission control prevents overbooking of the links. The router determines for every packet to which reservation it belongs and schedules it accordingly. There are no real deployments of this approach, due to scalability problems.

Differentiated services define a limited number of traffic classes and specify, per class, what service to expect from a router (PHB, per hop behavior). The packets are classified at the edge of the network, and the class is marked in the IP header. Routers in the interior of the network schedule packets according to that header. This scales way better than IntServ.

Week 8

Security

Security is an important aspect of network systems. Not everyone on the Internet can be trusted. However, historically, security was not important in the Internet. That is why there is no focus on security in the Internet's basic protocols, like TCP and IP. Security factors can be described as follows, with the descriptions explaining what would be possible without the factor:

- Confidentiality: can someone else read the messages between two people?
- Integrity: has someone modified a message you are receiving?
- Originality: can someone replay your actions, for example, paying twice for something?
- Timeliness: can someone delay a message for a long time?
- Authentication: is the resource you're using really the resource you intended to use?
- Access control: who can access a certain resource?

Symmetric encryption uses the same secret key to encrypt and decrypt data between two participants. The key must remain a secret. Without the key, deriving the plain text from the cipher text *should* be impossible. Examples of symmetric encryption algorithms are DES (deprecated), TripleDES (literally DES repeated three times) and AES (newer, favored).

Public key cryptography ensures confidentiality. Keys exist in pairs, where one key is used for encryption, and one for decryption. They can not be derived from each other. Public key cryptography can also be used to ensure authentication. Examples of public-key cryptography algorithms include RSA, Diffie-Helman (which only establishes keys) and elliptic curve cryptography. They all rely on a

certain mathematical difficulty: RSA on factorization, Diffie-Helman on discrete logarithms and elleptic curve cryptography on elleptic curves.

Symmetric algorithms are way faster than public key algorithms. They are often used in combination. The more secure public key algorithms are used to set up a temporary secret key, which is then used for symmetric encryption.

A cryptographic hash function is a trapdoor function, meaning that finding the input of the function based on its output is very difficult, if not impossible. A cryptographic hash function computes a fixed-length hash from an arbitrarily long input. The same input always yields the same output. Applications of a hash function include signing e-mails (hashing the e-mail's content using a secret key), testing the integrity of files (if the file has been tampered with, or if it's corrupt, the hash differs) and password storage. Hash functions are related to error detecting codes, where the integrity of the hash of a message is verified rather than the integrity of the message as a whole.

Hash functions need to be complex to be secure. If a hash function is simple, like CRC, then an attacker can simply feed possible inputs to the hash function repeatedly until the output matches the hash for which the attacker wants to know the plaintext input. Furthermore, hash functions should have a lot of possible outcomes, preventing collision attacks. It's not very difficult to alter a message without changing its CRC, making integrity verification useless.

Examples of hash functions include MD2, MD4, MD5 and SHA-1 (all deprecated, as attacks are now known) and SHA-2 and RIPEMD-160 (secure as far as we know so far).

A Message Authentication Code (MAC) is a fixed length value, calculated from a variable-length message and a fixed-length key. Without knowing the key, it's hard to calculate the MAC, or update the MAC after making a change to the message. It prevents forging or changing messages and provides authentication and integrity. It is similar to a digital signature, but based on symmetric cryptography.

Distributing keys is a difficult challenge, and we need to distinguish between short-lived session keys (like used in symmetric / public key hybrids) and longer-lived per-distributed keys (like public and private keys).

Session keys are only used during a single, relatively short episode of communication, and they are always used with symmetric cryptography, for speed. Every session gets a new key, and the session key is determined using a certain protocol. That protocol needs to be secure, and it uses the longer-lived pre-distributed keys. If a session key is found, the damage is limited, as only the data from a single session is revealed.

Fortunately, public keys do not need to be secret; anyone can share their public keys. However, authentication of the public keys is required, to ensure that the public key really belongs to its owner. Otherwise, a man-in-the-middle attack is possible, where the attacker uses its own public key imposing the actual participant of a conversation.

An unpractical approach is to simply meat the other party in real life and exchange the keys physically. However, you may also trust keys that have been personally verified by someone whose key you have personally verified, and so on.

An X.509 certificate contains a few fields, indicating the owner of the certificate, their public key, how long the certificate is valid, who crated the certificate and a hash of the other fields, encrypted with the issuer's private key. If you have the issuer's public key (which again can be personally verified), then you can verify the certificate. If you trust the issuer, you can also trust the certificate's subject's public key.

A public key infrastructure (PKI) can be organized in two ways. Either a hierarchy of certificate authorities trusting each other, or an informal web-of-trust. In the latter model, users exchange their certificates at real-life meetings, and users decide for themselves how much they trust the certificates.

In a key distribution for symmetric cryptography, keys are needed for any combination of two parties in the system. So for n parties, $n \cdot \frac{(n-1)}{2} \approx \frac{n^2}{2}$ keys are needed. If another (n + 1)th party joins, another n new keys are needed.

In a key distribution center, every party gets one secret key, known to them and the KDC. The KDC then generates session keys and encrypts them with the secret keys. Those session keys are used to communicate with other parties. Careful designs are needed to prevent replay and man-in-the-middle attacks. The advantage is that only n keys are needed for n parties, and a real secure channel is only needed once for each party. However, the KDC is a single point of failure; if it is compromised, all secret conversations become insecure. Furthermore, if a secret key with the KDC is compromised, all earlier communication of that party is also compromised.

The second lecture of week 8 includes a lot of examples of secure applications. I don't feel like summarizing them, just look through the slides.

Localization and synchronization

Fuck this shit.