Sieci Komputerowe

Notatki

„Modulo to, że mogę się zupełnie mylić, to działa to jakoś tak.”

Maciej Mikołajczak
Kraków, 2024/2025

Spis treści

1. Wprowadzenie

2024-10-03

Twierdzenie 1 (Fourier). Rozsądne funkcje okresowe wyrażają się szeregiem funkcji trygonometrycznych.

$$f(x) = c_0 + \sum_{i=1}^{\infty} a_i \sin(i \cdot x) + \sum_{i=1}^{\infty} b_i \cos(i \cdot x)$$

$$c_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)dx$$

$$a_i = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(i \cdot x)dx, \; b_i = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(i \cdot x)dx$$

Dowód. Trygonometria jest *magiczna*. □

Przedstawiamy tak w punktach które chcemy, w reszcie jest inaczej, ale to nam nie przeszkadza. Intuicja: najpierw rozkładamy pole pod wykresem na całą powierzchnię (c_0), potem przesuwamy funkcjami.

Uwaga. Losowy szum (mała funkcja losowa) ma małe współczynniki w rozkładzie Fouriera, więc mało zmienia przy przekazywaniu sygnału.

Twierdzenie 2 (Nyquist). Jeżeli funkcja f nie ma składowych o częstotliwościach większych niż B Hz i próbkujemy ją z częstotliwością $2B$ Hz, to możemy jednoznacznie odtworzyć f.

W szczególności więcej próbek nie ma sensu. Dlatego większa częstotliwość (np. światłowód) daje lepszą przepustowość niż mniejsza (kabel miedziany).

Wniosek. Maksymalna przepustowość to $2B \log \Sigma$.

Twierdzenie 3 (Shannon). Jeżeli $\frac{S}{N}$ to stosunek mocy sygnału do mocy szumu, to maksymalna przepustowość to $B \log(1 + \frac{S}{N})$, gdzie B jest najwyższą częstotliwością sygnału.

2. Warstwa łącza danych

2024-10-10

Model ISO-OSI:

- warstwa fizyczna
- warstwa łącza danych (głównie dalej poziom fizyczny, do tego np. rozwiązywanie problemów związanych z jednoczesnym nadawaniem przez różne urządzenia)
- warstwa sieci (tworzymy większe sieci za pomocą niższych warstw, które komunikują się na mniejszą skalę)
- warstwa transportowa
- warstwa sesji (ta i niższe warstwy umożliwiają programiście wykorzystywanie sieci)
- warstwa prezentacji
- warstwa aplikacji

Model TCP/IP:

- warstwa dostępu do sieci
- warstwa internetu
- warstwa transportowa
- warstwa aplikacji

Warstwa łącza danych (dostępu do sieci): chcemy jakiś system nadawania wiadomości, który umożliwi przekazywanie większych komunikatów. Chcemy umieć niwelować błędy w komunikacji, przede wszystkim wykrywać je, potem usuwać. Często ważna jest możliwość potwierdzenia komunikacji (tego, że odbierający słucha).

Synchronizacja zegara: chcemy wiedzieć, kiedy kończy się jeden komunikat i zaczyna drugi. Komunikujemy się ciągami 0 i 1, więc chcemy zrobić tak, żeby poprawna wiadomość nie miała za długiego ciągu takich samych znaków pod rząd – jeśli odbierany sygnał często się zmienia, to w miarę łatwo jest dostosować się do interwałów, w jakich kolejne sygnały są wysyłane, a gdy jest podtrzymywany ten sam sygnał to nie wiemy, ile razy go liczyć.

System Manchester: każdy bit kodujemy jako dwa znaki, duża strata przepustowości, ale prawie nie ma powtórzeń.

System NRZI (non-return-to-zero inverted): zaczynamy od wysyłania sygnału (czyli przekazywania 1), na wystąpieniu jedynki zmieniamy sygnał (z 1 na 0, z 0 na 1), a 0 nie zmienia sygnału. W takiej sytuacji złe stają się tylko długie ciągi 0, bo 1 zawsze oznacza zmianę.

Można to połączyć w system NRZI + 4B/5B: każde 4 bity zamieniamy na 5 za pomocą tabelki, w której maksymalny ciąg 0 jest krótki.

Używa się też systemu 8B/10B, który dodatkowo sprawia, że przekazywane wiadomości są DC-balanced. Problem (DC-bias) polega na tym, że jeśli odbiornik otrzyma za dużo jedynek (sygnałów z energią) na raz, to może to zmienić stan jego odczytu. Dlatego stosuje się takie ciągi dziesięciobitowe, które mają w miarę równą ilość 0 i 1.

Inny pomysł: random scrambling, czyli xorowanie wiadomości z pseudolosową liczbą i nadanie tego, odbierający xoruje z tym samym i odczytuje.

Historia Ethernetu: na początku kabel kocentryczmy, dopiero potem skrętka. Popularny, bo jest otwartą technologią, do tego łatwo go dostosować: można puścić po światłowodzie, kablach miedzianych, etc.

Nadawanie większych komunikatów Ethernetem:

- nagłówek (zwłaszcza w starych Ethernetach, 8 bajtów na zmianę 1 i 0 – ułatwia synchronizację)
- adres odbiorcy (adres MAC, 6 bajtów)
- adres nadawcy
- typ protokołu / długość komunikatu (2 bajty)
- dane (46-1500 bajtów)
- suma kontrolna (czyli hash, 4 bajty)

3. Wykrywanie błędów

2024-10-17

Parity bit – dodajemy na koniec bit mówiący, czy liczba jedynek jest nieparzysta (wiec ostatecznie w całej wiadomości jest parzyście wiele jedynek), daje około połowy szans na stwierdzenie, czy wiadomość jest z błędem

Odległość Hamminga – dla dwóch ciągów bitów tej samej długości jest to liczba różniących się bitów.

W celu wykrywania błędów będziemy przekształcać ciągi bitów długości 2^n na 2^m, gdzie $m > n$. Szukamy takiego przekształcenia $2^n \to 2^m$, w którym odległość Hamminga między wartością każdego argumentu to co najmniej $k+1$ – wtedy wykrywamy błąd na k bitach. Jeśli odległość między każdymi dwoma to co najmniej $2k+1$, to można odtworzyć nadany ciąg bez błędów – istnieje tylko jedna poprawna wiadomość, dla której mogła powstać taka wartość.

Ta tabelka musi być łatwo obliczalna i odwracalna, żeby to było praktyczne. Do tego chcemy szybko znajdować najbliższą poprawną wartość.

Przykład. Chcemy korygować 10 błędów. Jak duże musi być n?

Dowód. W każdej kuli (w sensie Hamminga) jest co najwyżej $\binom{m}{10} + \binom{m}{9} + \ldots \leq c \cdot m^{10}$ elementów. Zatem na pewno $2^m \geq 2^n \cdot cm^{10}$.

Haszowanie – haszujemy komunikat i doklejamy na koniec. Wykrywamy błąd porównując hash odebrany i wyliczony dla odebranych danych. Do korygowania możemy sprawdzać wszystkie możliwości (czy ich hash się zgadza z odebranym – jest to bardzo niepraktyczne, dlatego hashe zwykle służą tylko do wykrycia błędu).

CRC32 – rozszerzenie parity bita. Dla ciągu 2^n mamy wielomian nad $\mathbb{Z}_2$, gdzie współczynniki są takie jak wartości ciągu, np. $10010 \to x^4 + x$

Idea CRC32: dzielimy przez $x + 1$ i szukamy reszty. Sposób algorytmiczny: xorowanie (odejmowanie) każdej kolejnej pary bitów z naszego ciągu z 11. Resztą będzie parzystość jedynek w naszym ciągu, czyli parity bit.

W samym CRC32 będziemy rozważać wielomiany większe niż $x + 1$ i reszty z dzielenia przez nie. Dość dobrze działa $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$, on jest w Ethernecie (stąd nazwa hashowania – stopień wielomianu). Dla takiego wielomianu jesteśmy w stanie wykryć błędy do 3 bitów (zakładając ciąg bitów długości ramki Ethernet), dla większych błędów nie ma aż tak dużo pokrywających się wiadomości. Ethernet nie naprawia błędów, tylko je wykrywa i jest przy tym dość stabilny (bo jest na kablu), więc nie ma sensu robić silnego wykrywania i korekcji błędów.

Dzielenie kanału komunikacji. Nadawanie różnych komunikatów na raz powoduje, że nakładają się na siebie. Rozwiązaniem jest system (slotted) ALOHA.

Wysyłamy wiadomość, jak nie dostaniemy odpowiedzi, to był jakiś konflikt. Czekamy losową ilość czasu (żeby się znowu nie zderzyć) i powtarzamy. Jeśli słyszymy jakiś komunikat, to nie nadajemy – czekamy, aż będzie wolne. W kablu Ethernet to ma sens, ale przy sieciach radiowych niekoniecznie – możemy dostawać inne sygnały niż nasz odbiorca. Czas, jaki czekamy jest uzależniony od długości przekazywanych komunikatów (to jest właśnie *slotted* w nazwie systemu – każdy komunikat jest podobnie długi). Dlatego długość ramki Ethernet jest ustalona. Im dłużej nie udaje się nadać odebranego komunikatu, tym bardziej wydłużamy zakres z jakiego losujemy czas czekania (exponential back-off) – jak mamy konflikt z wieloma nadawcami, to małe czasy czekania nie wystarczą.

W Ethernecie wysłanie preambuły zajmuje tak długo, że każdy ją usłyszy. Dlatego jeśli każdy używa tego samego algorytmu, to po nadaniu preambuły jest pewność, że nikt inny nie nadaje.

W Ethernecie odbiorca jest ustalany po adresie MAC. Jest adres Broadcast – same jedynki, oznacza nadawanie do wszystkich.

Kabel miedziany umożliwia przesyłanie z częstotliwością ok. 125 MHz. Po zastosowaniu 4B/5B daje to ok. 100 Mb/s, czyli dobrze. W skrętce są 4 pary kabelków, bierzemy sobie parę i wysyłamy komunikację po niej, odbieramy po jakiejś innej. To powoduje, że nie ma problemów z dzieleniem kanału – mamy osobne kanały do nadawania i odbierania, a kabel łączy tylko 2 urządzenia, więc nie ma więcej problemów. Dwie pary kabelków pozostają nieużywane.

W Ethernecie właściwe możliwe są stany $-1, 0, 1$, bo napięcie może być w drugą stronę. W nowych Ethernetach stosuje się stany $-2, -1, 0, 1, 2$ (możemy puścić połowę napięcia). Gdyby puścić takie stany w jedną stronę po wszystkich kabelkach, to mamy około 1Gb/s. Mamy jednostronną komunikację, ale między dwoma urządzeniami – możemy jednocześnie nadawać i czytać, zależnie od tego, jakie jest odchylenie od naszego sygnału możemy stwierdzić, co powinniśmy odczytać. Daje nam do złącze gigabitowe po tych samych kabelkach. Jednocześnie daje nam to pewien rodzaj zabezpieczenia komunikacji – ciężko jest odtworzyć sygnał nie mając pewności, co nadaje jedna ze stron.

4. Warstwa sieci

2024-10-24

Numer MAC to numer fizyczny karty sieciowej, jest unikatowy, nowy nadawany każdej karcie, choć teoretycznie można zmienić sobie numer na poziomie software'u a systemy operacyjne same nadają adresy MAC wirtualnym kartom sieciowym, więc niekoniecznie są unikatowe.

Broadcast to adres MAC z samymi jedynkami. Służy do wysłania wiadomości do każdego. Karta sieciowa filtruje wiadomości i przekazuje do kernela te adresowane do niej i te broadcastowane. Można zmienić zachowanie karty sieciowej, by nie filtrowała.

Protokół ARP służy do tłumaczenia adresów IP na adresy MAC. Wysyłamy broadcast z pytaniem, czy jakaś maszyna ma dany adres IP, którego szukamy.

Sieci robią się duże, dlatego urządzenia są łączone switchami, które łączą się ze sobą dalej. Switche przekazują komunikaty (nie kopiują sygnału, tylko rozpoznają ramki i nadają je dalej – dzięki temu mogą same decydować o tym, kiedy wyślą wiadomość do każdego odbiorcy i np. robić współdzielenie łącza).

Algorytm switcha: nie odbija wiadomości (nie ma sensu przekazywać wiadomości maszynie, od której się ją dostało), broadcast wysyła wszędzie (poza źródłem). Ramka jest wysyłana w kierunku jej adresata (zapamiętuje, na jakich portach byli obserwowani dani nadawcy, potem wysyła do nich po tych portach) jeśli switch zna ten kierunek, inaczej do wszystkich. Takie ścieżki są pamiętane jakiś czas (rzędu minut), po zapomnieniu uczy się na nowo.

Problem pojawia się, gdy switche zaczną tworzyć cykl. Stosuje się Spanning Tree Protocol – switche uczą się sieci i eliminują cykle. W sieci wybierany jest lider (switch z najmniejszym MAC'iem), który na podstawie informacji o sieci decyduje, które kable nie będą używane.

VLANy – wirtualne sieci, mówimy switchom, po których kabelkach przesyłać daną komunikację, część maszyn traktować jak w jednej sieci, a część jak w drugiej. Połączenie pomiędzy switchami jest uznane za należące do obu sieci. W ramce trzymamy informację o tym, w jakiej sieci została wysłana. Zmieniamy ramkę tak, by było wiadomo, z której sieci idzie wiadomość. W polu z długością komunikatu trzymamy też tryb protokołu, wpisujemy długość dłuższą niż maksymalna możliwa, wtedy wiemy, że dalej będzie numer sieci wirtualnej i dopiero potem długość.

Większość z tych algorytmów działa na poziomie karty sieciowej, ARP w kernelu.

5. WiFi i sieci komórkowe

2024-10-30

Kanał komunikacji w sieciach WiFi: fale radiowe wysyłane w przestrzeń. W przeciwieństwie do kabli faktycznie występuje współdzielenie kanału komunikacji, do tego nie można stwierdzić, czy ktoś inny odbiera sygnał od kogoś innego – nadawca może być daleko od nas. Dlatego ciężko jest przeciwdziałać kolizjom. Błędy zdarzają się dużo częściej niż w Ethernecie, bo sygnał nie jest po kablu tylko leci przez świat.

Problemy z sieciami WiFi rozwiązuje się poprzez wprowadzenie punktów dostępowych (routery, przekaźniki), które mają sterować siecią. Access pointy łączą się osobnymi kanałami z każdym osobnym klientem, to pomaga w uniknięciu kolizji. Dzięki access pointom można łatwo łączyć sieci WiFi z sieciami Ethernet – można łączyć access pointy ze switchami Ethernetowymi, które mogą być połączone w większe sieci.

Urządzenia w sieciach WiFi są mobilne – regularnie się przemieszczają, zmieniają swoje access pointy. Do tego urządzenia mobilne z reguły mają mały dostęp do energii, więc trzeba ją oszczędzać.

W Ethernecie nie dbaliśmy o bezpieczeństwo, w sieciach komórkowych jest to dużo ważniejsze, bo przesyłany sygnał jest ogólnodostępny i łatwy do przechwycenia.

W sieciach WiFi sygnały słabną z odległością, występują zakłócenia (w szczególności sygnał może sam siebie zakłócić), często nie widać innych uczestników komunikacji – problem ukrytej stacji (nie widzimy, że ktoś inny nadaje do naszego rozmówcy, nie wiemy o zakłóceniu) i eksponowanej stacji (widzimy, że ktoś nadaje, a nasz rozmówca nie, wydaje nam się, że wystąpiło zakłócenie).

Metoda QAM (quadrature amplitude modulation) – przesyłanie informacji poprzez zmianę wysyłanej częstotliwości lub amplitudy (głośności), np. jeśli zobaczymy zakłócenia, to wyciszymy nasz sygnał.

Access pointy w WiFi muszą w jakiś sposób przekazywać informację o tym, że istnieją. Broadcastują taką informację co jakiś czas, dzięki temu nowe urządzenia wiedzą, jakie sieci WiFi są dostępne. Po połączeniu następuje uwierzytelnianie, potem powstaje połączenie kryptograficzne. Access point potrafi rozwiązywać konflikty między urządzeniami, daje im informację, czy wolno im nadawać. Dzięki temu obciążone sieci

są w stanie działać. Do tego access pointy przekazują sobie informacje przy zmianie access pointa przez urządzenie.

Błędy w sieciach WiFi są obsługiwane za pomocą kodów CRC (jak w Ethernecie), do tego pojawiają się też kody korygujące LDPC – heurystyka, która potrafi naprawić małe błędy. Potrzebujemy też potwierdzenia, że odbiorca dostał naszą wiadomość, inaczej nie wiemy, czy ramka została uszkodzona.

Aby współdzielić łącze uczestnicy sieci mogą wysłać do access pointa sygnały RTS (request to send), gdy chcą przekazać wiadomość. Wtedy access point może wysłać sygnał CTS (confirmation to send), który znaczy, że teraz będzie słuchał tego uczestnika komunikacji i inni nie powinni im przeszkadzać.

Szyfrowanie kluczem symetrycznym – metoda szyfrowania, kodujemy informację kluczem, potem odbiorca może odkodować informację tym samym kluczem. W tej chwili często stosuje się klucze AES. Szyfrujemy ustaloną liczbę bitów za pomocą klucza o ustalonej długości. Chcemy przesyłać dłuższe wiadomości niż nasze szyfrowanie szyfruje, możemy dzielić na bloki i szyfrować osobno, ale to może ułatwić złamanie szyfru – mamy lepsze metody.

W domowych sieciach WiFi często stosuje się uwierzytelnianie WPA-PSK (pre-shared key) – znamy hasło, na podstawie tego hasła generujemy klucz, którym szyfrujemy. Bierzemy hasło, losowe bity od access pointa i użytkownika, ich adresy MAC, to wszystko hashujemy i dostajemy klucz.

Następuje 4-way handshake:

1. Access point wysyła swoje losowe bity na początku nawiązywania połączenia.
2. Wtedy użytkownik ma już wszystko potrzebne bo szyfrowania, generuje klucz PTK, wysyła swoje losowe bity, a potem wysyła je zahashowane.
3. Wtedy access point może wygenerować klucz PTK i odkodować otrzymane bity. Jeśli się zgadza, to znaczy, że klient podał dobre hasło. Access point wysyła klucz GTK (jeden, znany przez wszystkich użytkowników sieci, wykorzystywany do broadcastowania) zaszyfrowany kluczem PTK.
4. Użytkownik potwierdza nawiązanie połączenia.

Dzięki dodaniu losowych bitów do klucza mamy pewność, że klucz zmieni się przy każdym nawiązaniu połączenia, czyli dość często. Dzięki temu długie zbieranie danych nie pomoże w odszyfrowaniu komunikacji.

Każdy inny użytkownik sieci może podsłuchać wysyłane losowe bity i zna hasło, więc może wygenerować klucz PTK związany z komunikacją innego użytkownika. Dlatego nie można zakładać, że osoby z wnętrza sieci nie widzą komunikacji. W nowszych WiFi używa się WPA3 – lepszy schemat generowania klucza, chroni też przed innymi uczestnikami sieci.

Kiedyś było WPA1 – klucz tylko na podstawie hasła, wtedy bardzo dużo informacji szyfrowane tym samym kluczem (nie zmienia się bardzo długo), daje więcej informacji atakującemu.

Aby WiFi było kompatybilne z Ethernetem stosuje się te same adresy MAC. Do tego ramki muszą być między sobą konwertowalne. Ramki WiFi są dłuższe niż Ethernet (bo muszą przekazać dużo więcej informacji), ale są tak zbudowane, że można po prostu wsadzić ramkę Ethernet w ramkę WiFi. W szczególności ramki WiFi mają 4 pola na adres – adresy nadawcy i odbiorcy faktycznej wiadomości oraz adresy access pointów, które przesyłają sobie wiadomość (w komunikacji radiowej jest to potrzebne).

Aby oszczędzać energię klienta sieci, przez większość czasu wyłącza się kartę sieciową. Aby jednak brać udział w komunikacji karta sieciowa włącza się co jakiś czas. W tym celu przekazuje się informację o tym access pointowi, który buforuje ramki adresowane do danego klienta aż on włączy swoją kartę sieciową i będzie mógł mu je przekazać. Wraz z broadcastem informującym o jego adresie MAC access point wysyła też adresy MAC klientów, dla których ma ramki. Wtedy klient wie, że powinien włączyć kartę sieciową i odebrać te ramki.

6. Sieć IP

2024-11-07

Rozważamy warstwę internetu, czyli tworzenie sieci, które są bardzo duże. Potrzebujemy w jakiś sposób adresować nasze maszyny, mieć sposób znajdowania tras dla naszych pakietów. Małe sieci, które będziemy łączyć, będą bardzo różne, kolejne metody transportu danych będą się różnić, musimy mieć komunikat,

który będzie pasował do wszystkich. Chcielibyśmy też mieć możliwość wysyłania do wielu adresatów, w jakiś sposób związanych ze sobą. Musimy też zajmować się buforowaniem danych.

Na niższych poziomach mamy nadane losowo adresy MAC, pojawiają się też różne dziwne pomysły (np. adresowanie sieci za pomocą klucza publicznego, którego używa kryptografia sieci).

Mamy adresy IP – krótkie (4 bajty), prefix adresu ma sygnalizować lokalizację – adresy o tym samym prefixie powinny znajdować się blisko siebie (geograficznie). Mamy też adresy DNS – adresujemy czytelnym dla człowieka ciągiem znaków.

Komputery łączące się w sieć są ze sobą bezpośrednio połączone za pomocą różnych technologii. Jeden komputer generuje pakiet IP i adresuje go do innego komputera. Chcemy znaleźć ścieżkę między nimi w sieci, która jest jak najkrótsza i daje najmniejsze opóźnienie. Jednocześnie chcemy to robić szybko. Właściwie sprowadza się to do znalezienia najkrótszej ważonej ścieżki w grafie. Problem jest taki, że sieć cały czas się zmienia, bo np. jedne połączenia są bardziej zajęte niż inne. Do tego komputery mogą łączyć się i odłączać.

Naszą sieć mogą modelować sieci przepływowe, gdzie pakiety płyną ze źródeł do ujść. Przy ustalonych ujściach dla każdego źródła (multi commodity flow) problem jest NP-trudny, więc nie da się tego zrobić optymalnie w sensownym czasie. Podobnie trudne jest stwierdzenie, które pakiety należy w danej chwili przesłać po jakich krawędziach mając zadaną pełną listę komunikatów.

Pomysł na efektywny routing jest taki, że tworzymy hierarchię sieci, dzielimy ją na fragmenty, które odpowiadają prefixom w numerach IP. Znacznie ogranicza to rozmiar problemu – jeśli dzielimy po bajtach, to na najwyższym poziomie mamy 256 wierzchołków, więc nawet trzymanie wszystkich tras działa sensownie. Następnie wewnątrz tych fragmentów znowu dzielimy na fragmenty i powtarzamy cały pomysł.

W tej chwili dzielimy nie na cztery jednostki, tylko na dwie. Mamy Autonomous Systems (AS), które odpowiadają pierwszej połowie adresu IP. Wewnątrz nich rozchodzimy się na faktyczne adresy IP maszyn.

Podczas przesyłania danych pojawia się problem, gdy wiele pakietów chce wykorzystać to samo połączenie. Wtedy nie mamy miejsca, żeby zmieścić wszystkie na raz, musimy buforować pakiety. Dane są przesyłane po równo (wolniej niż przychodzą), reszta jest zapamiętywana, czeka w kolejce do wysłania. Jeśli bufor jest nieograniczony, to wraz ze wzrostem kolejki opóźnienia mogą bardzo rosnąć.

Dlatego ograniczamy bufor, jak przychodzi niemieszczący się pakiet, to możemy wyrzucać ten, który jest najstarszy – wtedy mamy zerową przepustowość, bo nawet jak wyślemy pakiet, to następny router go odrzuci, bo już będzie stary. Możemy wyrzucać losowe pakiety, co powoduje, że połączenia wysyłające więcej są karane (bo mają większą szansę, że ich pakiet zostanie odrzucony). To powoduje, że komputery będą chciały dzielić się połączeniami po równo.

Nagłówek IP ma (co najmniej) 20 bajtów, mamy:

- 4 bity na numer wersji
- pole IHL, które steruje tym, że można wydłużyć nagłówek
- dwa bity ECN (early congestion notification) zapalane, gdy następuje buforowanie pakietów
- długość pakietu na dwóch bajtach (więc rozmiar pakietu nie większy niż 2^{16}, a właściwie nawet mniejszy)
- 13 bitów na fragment offset – jaki jest offset tego fragmentu od początku całej wiadomości,
- Time To Live (TTL) na jednym bajcie, wartość, którą każdy router zmniejsza przy przesyłaniu i po dojściu do 0 odrzuca pakiet (zapobiega to cyklom)
- protocol na jednym bajcie, który mówi, jaki protokół jest pod spodem (np. UDP, TCP)
- checksuma na 2 bajtach
- po 4 bajty na adres źródła i celu
- dodatkowe miejsce zależne od IHL

Pakiety IP podczas przesyłania mogą być dzielone na części, bo nie każda technologia może przesyłać ramki o takiej samej wielkości. Przy przesyłaniu ich np. Ethernetem tworzymy ramkę Ethernet, w której dane zaczynają się nagłówkiem IP i potem następują dane tego pakietu (ich fragment). Protokół IP

zajmuje się tym, żeby podzielić swój pakiet na części. Odbiorca czeka, aż dostanie cały pakiet, zanim zacznie go przekazywać.

Ważne adresy IP, wydzielone do specjalnych zadań (notacja /x oznacza, że pierwsze x bitów jest ustalonych):

- 0.0.0.0/8 – obecna sieć
- 127.0.0.0/8 – adres loopback, w szczególności 127.0.0.1 to localhost
- prywatne sieci 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, które są używane w sieciach lokalnych
- 255.255.255.255 – broadcast

Adresy IP wewnątrz prywatnej sieci są stosowane tylko w niej, nie da się zaadresować bezpośrednio maszyny w innej sieci prywatnej. Jeśli chcemy ją zaadresować, to robimy to przez pośrednika, czyli główny router w sieci (protokół Network Address Translation, NAT) – jeśli chcemy wysłać pakiet z sieci prywatnej, to router adresuje jakby był od niego. Jak coś przychodzi, to jest adresowane do routera. Router po otrzymaniu pakietu znowu go przeadresowuje. W komunikacji występuje numer portu, który router zmienia tak, aby wiedział, do której maszyny lokalnej przesłać pakiet.

Jeśli dwie maszyny o adresach prywatnych chcą się ze sobą skomunikować, to potrzebują do tego wykorzystać osobną maszynę, która ma adres publiczny. Ta maszyna może "zgadnąć" jak routery przydzielają porty i przekazać maszynom, na jakie porty powinny pisać. Dzięki temu będą mogły się komunikować bezpośrednio.

7. Routing w sieciach IP

2024-12-12

W danej maszynie mamy różne reguły routingu (zbiory adresów IP i maszyny, do których będziemy je wysyłać). Reguły są typu: 0.0.0.0/0 przesyłamy do routera, 127.0.0.0/8 przesyłamy do 127.0.0.1 (loopback naszej maszyny) i inne podobne. Jeśli mamy kilka pasujących wzorców, to wybieramy ten najkonkretniejszy (z największą ilością ustalonych bitów).

W małych sieciach routing adresów IP może odbywać się właściwie dowolnie (tak, jak chce jej administrator). Wszystkie metody routingu nazywamy IGP (Internal Gateway Protocol).

Tablice routingu często są wyznaczane za pomocą RIP (Routing Information Protocol), który wyznacza (heurystycznie) najkrótsze ścieżki i przekształca je na tablice routingu. Ten protokół jest oparty o serwery, które komunikują sobie, jakie połączenia widzą, każdy tworzy distance vector – najmniejsze odległości do wszystkich innych. W każdym kroku wysyłamy swój distance vector do sąsiadów, na podstawie otrzymanej informacji poznajemy odległość do nowych serwerów, zapamiętujemy ją.

Ten algorytm jest stosowany w małych sieciach, bo wymaga liniowego czasu i pamięci. W przeciwieństwie do Ethernetowych switchy korzystamy z cykli – każdy wierzchołek widzi graf jako drzewo, ale każdy jako inne. Do tego możemy usuwać krawędzie – po pozbyciu się jakiejś protokół dalej działa, wyznaczy nową najkrótszą ścieżkę. Przez chwilę wartości w wierzchołkach będą niewłaściwe, ale z każdym krokiem będą zbiegać do tych poprawnych.

Drugim rozwiązaniem jest OSPF (Open Shortest Path First), który przesyła pełne najkrótsze ścieżki, a nie tylko distance vector – Link State.

Pomiędzy ASami musimy stosować ten sam protokół EGP (Exterior Gateway Protocol), czyli protokół przesyłania między dużymi sieciami. Jego przykładem jest BGP (Border Gateway Protocol), który wyznacza najkrótsze ścieżki na podstawie informacji od innych wierzchołków. Każdy AS przesyła innym długość proponowanej ścieżki i kolejny adres, do którego pójdzie komunikacja. Dzięki temu podczas wysyłania wiadomości mamy trochę większą kontrolę niż w RIP.

Do tego wszystkiego mamy protokoły pomocnicze:

- ARP – zdobywanie adresów MAC na podstawie adresów IP
- DHCP – automatyczna konfiguracja adresów IP; nowy członek sieci broadcastuje prośbę o konfigurację, serwer DHCP wysyła do niego przyznany adres IP i kilka innych ważnych adresów (broadcast, gateway)

- ICMP – ruch, z którego korzystają narzędzia do debugowania ruchu IP, np. ping, mtr (w ICMP odrzucenie pakietu, któremu skończył się TTL skutkuje wysłaniem nadawcy informacji o tym)
- IGMP – wysyłanie jednej wiadomości do wielu adresatów (idzie jedną ścieżką tak długo, jak się da, potem się rozdziela) lub do jednego z wielu adresatów; stosuje się to w sieciach lokalnych, w dużych nie ma sensu

8. UDP i TCP

2024-12-19

Datagram UDP to właściwie pakiet IP z dopisanymi numerami portów.

TCP to próba uzyskania niezawodnego połączenia. Chcemy nie tracić przekazywanych pakietów – stosujemy potwierdzenia transmisji i retransmisje w przypadku ich braku. Zakładamy, że poprawne potwierdzenie jest poprawne (a nie jest zepsutym potwierdzeniem czegoś innego). W takiej sytuacji jeśli dostaniemy potwierdzenie, to na pewno nasz pakiet dotarł.

Potwierdzenia w TCP wymagają większego czasu, ale dają pewność komunikacji. Dlatego bardziej niskopoziomowa komunikacja (np. DNS) idzie po UDP. Do tego nie istnieje broadcast w TCP – nawiązujemy połączenie z danym klientem i się z nim komunikujemy.

TCP umożliwia nawiązywanie i utrzymywanie połączenia oraz zapewnia retransmisję. Potrafi też kontrolować czas transmisji i rozmiar buforów – jeśli odbiorca nie nadąża z czytaniem i musi buforować, to może poprosić nadawcę o wstrzymanie wysyłania.

Połączenie TCP jest podtrzymywane, dopóki nie zostanie zamknięte. Wtedy następuje wymiana oznaczająca zgodę na zamknięcie połączenia i potwierdzenie, że wszystkie wysłane pakiety zostały odebrane.

TFTP – proste przesyłanie plików, podobne do TCP, jest pojęcie sliding window – liczba pakietów, które można wysłać, zanim będziemy oczekiwać na potwierdzenie. Odbiorca potwierdza całe bloki, odsyła potwierdzenie ostatniego, jaki dostał (po całym bloku lub po niedostaniu któregoś). Jeśli dostaniemy pakiet późniejszy niż jakiś, którego nie mamy, to zgłaszamy niedostanie tego pakietu. Podobnie działa TCP.

W nagłówku TCP mamy sequence number – numer aktualnego pakietu. Ma on 32 bity. Gdy nam się skończą, to zapętlamy ten numer (dzieje się to co dużą liczbę pakietów, więc nie jest problemem). Mamy też acknowledgement number – ile pakietów odebraliśmy (a raczej o jeden więcej – numer pakietu, na który jesteśmy gotowi). Jedna wiadomość może przesyłać jakieś dane i potwierdzać poprzednią komunikację. Do tego właśnie służy ta liczba. Jest pole na długość nagłówka (mogą wystąpić dodatkowe opcje wydłużające nagłówek).

W TCP długość sliding window jest zależna od tego, jak szybkie połączenie chcemy. Jest zmieniana w zależności od stanu łącza. Jest też metoda selective repeat – prosimy tylko o pakiety, które się zgubiły. Druga metoda jest lepsza, gdy mało pakietów się gubi – nie trzeba przesyłać całego okna. Pierwsza jest częściej stosowana, bo działa lepiej, gdy komunikacja przebiega bezproblemowo.

Uznanie pakietu za zagubiony dzieje się, gdy nie przychodzi wystarczająco długo – jest to wartość zależna od wyliczonego RTT (round trip time). Dokładny wzór jest podany w odpowiednim RFC – w jakiś sposób wyznaczamy średnie RTT i jego odchylenie standardowe.

Zgłoszenie uszkodzonego pakietu to wysłanie kilku potwierdzeń poprzedniego pakietu ciągiem.

Gdy klient TCP chce zacząć połączenie wysyła wiadomość do słuchającego serwera. W tej wiadomości ustala początkową wartość sequence number (nie zaczynamy zawsze od 0). Dzięki temu gdy zaczynamy nowe połączenie z tymi samymi danymi (z tej samej maszyny i portu) serwer wie, że jest to nowe połączenie. Serwer odsyła potwierdzenie i swój sequence number. Klient potwierdza nawiązanie połączenia. W tym momencie serwer faktycznie tworzy połączenie i zaczyna się komunikacja.

Wysyłane dane są buforowane w kernelu, bo lepiej jest wysyłać duże pakiety (nie marnować pakietu na przesłanie małej ilości bajtów). Możemy zpushować (zflushować), jeśli wiemy, że chcemy wysłać mało bajtów. W nagłówku jest pole URG, które służy do wysyłania priorytetowych segmentów. Nie jest to często stosowane.

Pakiety ACK (potwierdzenia) staramy się wysyłać od razu – jeśli poczekamy na kolejny pakiet, żeby potwierdzić zbiorczo, to on może nie przyjść. Przyspiesza się na inne sposoby, które są zdefiniowane w

dodatkowych RFC. Stosując odpowiednie opcje nagłówka można zastosować selective repeat (by dostać konkretne zagubione pakiety), co się przydaje przy dużym oknie. Jest też miejsce na wpisanie RTT, co pomaga w liczeniu odpowiedniego timeoutu.

Są różne pomysły ustalania prędkości przesyłania. Jednym z nich jest TCP Tahoe. Zaczyna się od małej wartości, rośnie wykładniczo do ustalonej stałej, potem liniowo. Przy pojawieniu się problemów prędkość spada do początkowej wartości. TCP Reno robi w miarę to samo, tylko spada do tej wartości, od której zaczyna się wzrost liniowy. Inne pomysły zamieniają wzrost liniowy na coś innego. W Linuxie domyślny jest Cubic, który używa funkcji sześciennych. Działają też mechanizmy, które dostosowują prędkość do innych połączeń, które akurat działają w systemie.

9. HTTP

2025-01-09

HTTP to taki system plików, w którym pliki adresujemy URLami. Podajemy protokół, serwer i ścieżkę.

HTTP/0.9 to najprostsza wersja HTTP, w niej jest tylko request ze ścieżką, nie ma nagłówków.

W HTTP/1.0 mamy typ requestu, ścieżkę, wersję i nagłówki – pary jak w słowniku. Serwer zwraca typ odpowiedzi, nagłówki i zawartość pliku.

Typy requestów:

- GET – zwykłe pobieranie pliku
- HEAD – dostajemy tylko nagłówki
- POST – umożliwia wysłanie pliku, po nagłówkach pusta linia i dalej wysyłany plik

Mamy sporo możliwych nagłówków:

- Date – przekazanie czasu
- Authorization – autoryzacja użytkownika
- From – od kogo
- If-Modified-Since – chcemy plik tylko, jeśli się zmienił, przydatne do cachowania
- Location – serwer nakierowuje klienta w inne miejsce
- Server – wersja oprogramowania na serwerze
- WWW-Authenticate – autoryzacja od strony serwera
- Expires, Last-Modified – związane z cachowaniem
- Content-Type – dane o pliku

W HTTP/1.1 mamy trochę zmian, pojawia się nagłówek Connection, wartość keep-alive – będziemy chcieli kolejne pliki, ściągniemy je w jednym połączeniu TCP. Dzięki temu nie musimy go nawiązywać od nowa. Minus jest taki, że musimy faktycznie przekazywać rozmiary plików (nowy nagłówek Content-Length), a serwer musi trzymać połączenie nawet jak klient aktualnie o nic nie prosi. Jeśli chcemy skończyć połączenie przekazujemy close.

Mamy nowe nagłówki: Host (często jeden serwer odpowiada za wiele domen, serwer musi rozróżniać, jaką domenę chciał klient, więc ją doklejamy do requestu), ciasteczka, kompresje, utrzymywanie połączeń, range (danie części pliku)

W HTTP/2 serwer może wysyłać odpowiedzi w innej kolejności niż dostał zapytania lub nawet przesyłać je w kawałkach, które się przeplatają. HTTP/3 zrezygnowało z TCP, ma swój protokół oparty o UDP.

Accept-Language – preferowany język pliku, który ma zwrócić serwer. Accept – to samo z typem pliku.

Długość pliku możemy przekazać dając Transfer-Encoding: chunked (serwer może nie znać długości pliku od razu), dane zaczynają się od rozmiaru chunku i kończą na rozmiarze następnego. Ma to sens np. gdy serwer generuje odpowiedź za pomocą jakiegoś programu.

Serwer może przekazać nagłówek ETag – identyfikator pliku, służy do cachowania, możemy poprosić o plik, jeśli nie pasuje do etaga (czyli się zmienił) – nagłówek If-None-Match.

Ciasteczka: serwer może dać nagłówek Set-Cookie, tam ciasteczko, które klient ma wstawiać w swoje requesty – serwer pamięta jakieś dane po tym ciasteczku.

10. SSL

2025-01-22

Chcemy, żeby przesyłane wiadomości były poufne (nie da się przechwycić), integralne (nie da się zmienić, w szczególności skrócić lub wydłużyć). Interesuje nas też tożsamość komunikujących się (chcemy potwierdzić tożsamość lub na odwrót – schować ją).

Potrzebujemy funkcji haszujących, które są losowe i nieodwracalne. Jest dużo takich funkcji – SHA, MD4, MD5. Wszystkie te funkcje są już stare i nie stosuje się ich. Wszystkie one działają właściwie na podobnej zasadzie, tylko są coraz bardziej skomplikowane.

W MD4 jest haszer, który ma stan. Input zostaje podzielony na bloki stałego rozmiaru (takiego jak stan). Na początku jest jakiś ustalony stan bazowy. Robimy padding inputu do pełnego bloku (nie można zerami, bo wtedy sporo wiadomości ma ten sam hasz, trzeba zapisać tam rozmiar wiadomości). MD4 przepuszcza kolejne bloki przez stan – daje to nowy stan, ostatni stan to wynik. Przepuszczanie to dość skomplikowana operacja, która miesza ze sobą bity w taki sposób, żeby wynik mocno zależał od każdego bitu.

Szyfrowanie symetryczne – jest jakiś klucz i osoby, które go mają, potrafią kodować i dekodować wiadomości. Tutaj potrzebujemy odwracalności, ale tylko jeśli znamy klucz. Takie szyfry to np. szyfr Cezara czy xorowanie przez ustaloną liczbę. Z xorowaniem jest taki problem, że jak ktoś zgadnie wiadomość, to odzyska z tego klucz. Ale jest to dobry pomysł, gdy wiadomości są nieprzewidywalne. Współcześnie stosuje się głównie szyfrowanie AES.

AES polega na przechodzeniu między stanami i shufflowaniu wiadomości (podobnie jak w MD4), ale te operacje są odwracalne gdy znamy klucz. Jesteśmy w stanie szyfrować bloki długości takiej jak klucz. Możemy kolejne bloki szyfrować niezależnie – zły pomysł, bo poważające się bloki wiadomości dają powtarzające się bloki zaszyfrowane. Rozwiązanie tego to CBC – zaczynamy od jakichś ustalonych losowych bitów, xorujemy i dopiero kodujemmy, potem jako losowe bity podstawiamy wynik dla poprzedniego bloku. Prostszym rozwiązaniem jest CTR – mając losowe bity v szyfrujemy je i xorujemy przez wiadomość, potem inkrementujemy v. CTR jest o tyle dobre, że da się je zrównoleglić.

Do ustalenia wspólnego klucza używa się protokołu Diffie'go-Hellmana. Jedna strona chce przesłać drugiej a, druga pierwszej b. Mamy ustalone liczby pierwsze g, p. Teraz $g^a, g^b \pmod p$ są nieodwracalne, obie strony mogą policzyć g^{ab}, ale nikt nie zna obu a i b – tak ustalamy klucz. To działa, o ile nikt nie zmienia przesyłanych komunikatów. Wtedy ktoś inny może zrobić to samo z liczbą c i przesłać g^c obu stronom – wtedy one stworzą klucz korzystający z c i pośrednik będzie mógł zmieniać komunikację. Takiej sytuacji (Man in the Middle Attack) ciężko uniknąć. Dlatego ważne są certyfikaty SSL – wiemy dokładnie z kim rozmawiamy.

Mamy protokół Needhama-Schroedera. Istnieje centralne S, z którym A i B mają już wspólny klucz. Wtedy A prosi S o nawiązanie komunikacji. S tworzy klucz K_{AB}, wysyła go A i zakodowane bity $\text{enc}_{K_{BS}}(K_{AB}, A)$. Wtedy A może przesłać je B i B będzie wiedzieć, że ta wiadomość to poprawny klucz komunikacji z A. Na koniec B wysyła A jakąś liczbę, a A odsyła ją zdekrementowaną – dzięki temu potwierdzamy, że klucz jest taki sam po obu stronach.

Taki protokół ma jedną słabość – jeśli ktoś podsłucha $\text{enc}_{K_{BS}}(K_{AB}, A)$, to może przekonać B, żeby znów korzystać z tego klucza, co jest problemem, jeśli K_{AB} wycieknie. Dlatego dodatkowo przesyła się czas, klucz działa tylko przez chwilę. Wtedy trzeba pamiętać o synchronizacji czasu. Takie coś działa np. w Kerberosie.

Szyfrowanie kluczem publicznym, np. RSA. Znowu potęgujemy liczby modulo i korzystamy z tego, że łatwo jest odwrócić mnożenie jak wie się co było mnożone. Podpisywanie wiadomości m to po prostu dopisanie do niej $\text{dec}(h(m))$, gdzie h jest funkcją haszującą a $\text{dec}(\cdot)$ to kodowanie kluczem prywatnym.

Szyfrowanie SSL polega na przedstawieniu kluczy publicznych i certyfikatów – jedna strona je daje, podpisuje swoją część sekretu, wtedy nikt pomiędzy nie może przedstawić swojego sekretu zamiast tego i nie będzie w stanie przechwytywać komunikacji.

SSL działa na najwyższym etapie komunikacji – pomiędzy TCP a HTTP. Można zrobić to samo na niższym poziomie, np. IPsec działa na poziomie IP. Potrzebujemy systemu podobnego do certyfikatów, tylko dla adresów IP. Można ręcznie wgrywać certyfikaty w system. Można też stosować do tego DNS, ale on nie jest wiarygodny – nie jest szyfrowany. Dlatego wprowadzamy pomysł podpisywania rekordów DNS, mamy root servery, więc naturalną hierarchię dziedziczenia zaufania.

Możemy używać serwerów VPN – proxy, do którego idzie nasz ruch. On jest szyfrowany, potem idzie dalej. Jesteśmy bezpieczniejsi, bo komunikujemy się bezpośrednio z jednym serwerem po zabezpieczonym połączeniu. VPN może działać na różnych poziomach – zazwyczaj na TCP, ale możemy stworzyć połączenie na niższych poziomach (np. w Ethernecie, tworząc sztuczną kartę sieciową).

11. Sieci P2P

2025-01-23

Chcemy stworzyć zdecentralizowaną sieć, która będzie niezależna od działania konkretnych serwerów – protokół BitTorrent.

Mamy wiele klientów, żadnego serwera, pobieramy pliki od każdego – wysyłamy zapytanie o plik, dostajemy odpowiedź od dowolnego klienta. Jest podział na paczki, każdą paczkę można dostać od kogoś innego. Jest zachęta do wysyłania plików w sieci – klienci wysyłają dane tylko do klientów, którzy im coś wysłali. Z tego powodu klienci faktycznie dzielą się plikami, a nie tylko pobierają je.

Dane trzymane są w Distributed Hash Table. Klienci i dane mają identyfikatory będące haszami ustalonej długości. Klienci pamiętają części bazy – od swojego hasza do następnego hasza jakiegoś klienta. Zamiast pamiętać wszystkich klientów robimy jump pointery – pamiętamy adres pierwszego, drugiego, czwartego, ósmego i tak dalej klienta od nas, możemy się komunikować w czasie logarytmicznym.

Każdy klient pamięta część tablicy, a że mogą się komunikować, to można przekazać prośbę o odczyt lub zapis. W ten sposób pamiętamy dane, którymi się dzielimy i tablice klientów, którzy posiadają dane. Taki system nazywa się Chord – jest prosty koncepcyjnie, ale ciężko go zastosować w praktyce, dlatego BitTorrent obi coś innego.

Klienci mogą się odłączać, chcemy nie stracić naszej bazy. Dlatego trzymamy dane w miejscu odpowiadającemu ich haszowi i jeszcze w kilku kolejnych. Nowi klienci muszą mieć połączenie do jakiegoś jednego, mając to mogą się wpiąć w bazę – nadać sobie hasz i zająć odpowiednie miejsce. Takie zmiany powodują, że jump pointery często się deaktualizują, więc co jakiś czas budujemy je od nowa.

Można przejąć bazę jak poda się konkretny hasz i połączy wiele swoich klientów z danym obszarem bazy – wtedy ma się wyłączne posiadanie części danych i można je usunąć odłączając się. Dlatego chcemy aby obliczenie hasza klienta było trudne obliczeniowo (nie można wtedy samemu stworzyć wielu klientów) i co jakiś czas zmieniamy haszowanie.

Wpisując plik do tablicy wpisujemy go pod jego hasz – wtedy sam się podpisuje i sprawdzenie poprawności to po prostu policzenie hasza i porównanie z miejscem w tablicy.