**რატომ შეიქმნა OSI მოდელი?**

1980-იან წლებში, როდესაც ინტერნეტის წინამორბედი და სხვადასხვა ქსელური სისტემები ვითარდებოდა, სხვადასხვა მწარმოებელი იყენებდა საკუთარ არათავსებად პროტოკოლებსა და სისტემებს. მაგალითად, IBM-ს, Microsoft-ს, Apple-ს და სხვა კომპანიებს ჰქონდათ განსხვავებული მიდგომები ქსელურ კომუნიკაციაზე. ეს ქმნიდა პრობლემას: ორი მოწყობილობა სხვადასხვა მწარმოებლიდან ხშირად ვერ “ესაუბრებოდა” ერთმანეთს, რადგან არ არსებობდა უნივერსალური ენა, რომელსაც ორივე მხარე გაუგებდა.

ISO (International Organization for Standardization) ცდილობდა შეექმნა უნივერსალური ჩარჩო, რომელიც შეძლებდა სხვადასხვა მოწყობილობას შორის კომუნიკაციის სტანდარტიზაციას. ასე შეიქმნა OSI მოდელი – *კონცეფცია*, რომელიც არ არის კონკრეტული პროტოკოლი, არამედ გეგმა, რომელიც ხსნის, როგორ უნდა გაიყოფოს მონაცემთა გადაცემა 7 განსხვავებულ, ლოგიკურ ფენად.

OSI მოდელი დღესაც გამოიყენება როგორც საგანმანათლებლო და ანალიტიკური ინსტრუმენტი – იმის გასაგებად, სად შეიძლება იყოს ხარვეზი ქსელში, ან როგორ მუშაობს კომუნიკაცია კონკრეტულ ეტაპზე.

**7. Application Layer**  
ყველაზე ზედა ფენა, რომელიც ახლოს არის მომხმარებელთან. აქ ხდება ვებსაიტის გახსნა, ელფოსტის გაგზავნა ან ფაილის გადმოწერა. იყენებს პროტოკოლებს, როგორიცაა HTTP, FTP და DNS. ეს ფენა უშუალოდ უზრუნველყოფს სერვისს, რასაც მომხმარებელი იყენებს.

**6. Presentation Layer**

* ასრულებს მონაცემების გარდაქმნას ისეთ ფორმატში, რომელიც მარტივად აღსაქმელია აპლიკაციის ფენისთვის
* უზრუნველყოფს მონაცემების დაშიფვრასა და გაშიფვრას გადაცემისა და მიღების პროცესში
* ახორციელებს მონაცემების შეკუმშვას, რათა ისინი უფრო სწრაფად და ეფექტიანად გადაეცეს სესიის ფენას

**5. Session Layer**  
ეს ფენა ამყარებს "სესიას" – ანუ კავშირს ორ მოწყობილობას შორის. იწყებს, ინარჩუნებს და ასრულებს კავშირს. მაგალითია ვიდეოზარი ან ჩატი. იცის, ვინ როდის ლაპარაკობს და როდის უნდა გაჩერდეს.

**4. Transport Layer**  
ამ ფენაზე ინფორმაცია იყოფა პატარა ნაწილებად – სეგმენტებად. ამოწმებს, სწორად და სრულად მივიდა თუ არა ყველაფერი. იყენებს TCP ან UDP პროტოკოლს. ასევე უკავშირდება კონკრეტულ პროგრამებს პორტების მეშვეობით.

**3. Network Layer**  
აქ უკვე ჩნდება IP მისამართი – კომპიუტერის ციფრული მისამართი. ეს ფენა არჩევს საუკეთესო მარშრუტს მონაცემისთვის. პაკეტებად ყოფს მონაცემს და აგზავნის სხვა ქსელებშიც. როუტერები მუშაობენ ამ ფენაზე.

**2. Data Link Layer**  
ეს ფენა ქმნის ფრეიმებს და ამოწმებს, სწორად გავიდა თუ არა ინფორმაცია. იყენებს MAC მისამართებს მონაცემის ადრესატამდე მისაყვანად. ასევე არ უშვებს ერთდროულად გაგზავნას ორ მოწყობილობას შორის . შეცდომის აღმოჩენაც ამ ფენაზე ხდება.ეს ფენა პასუხისმგებელია ორი მეზობელი მოწყობილობის (დამაკავშირებელი მოწყობილობები ერთსა და იმავე ქსელში) შორის მონაცემთა გადაცემის სანდოობაზე.

**1. Physical Layer**  
**OSI მოდელის პირველი ფენა,** რომელიც **ზრუნავს იმაზე, როგორ გადაეცეს 0 და 1 ფიზიკურად,** ანუ **როგორ ვაქციოთ ციფრული ინფორმაცია ელექტრულ, რადიო ან ოპტიკურ სიგნალებად** და პირიქით.

**სლაიდი 3**

**აპლიკაციის ფენა** - არის კომპიუტერული ქსელის ზედა ფენა, რომელიც პასუხისმგებელია იმაზე, რომ მომხმარებლები (მაგალითად, თქვენ) შეძლონ პროგრამების გამოყენება და ინფორმაციის გაცვლა ინტერნეტში. ეს ფენა უზრუნველყოფს, რომ თქვენმა მოთხოვნამ (მაგ., ვებგვერდის გახსნა) მიაღწიოს სერვერს და მიიღოთ პასუხი სასურველ ფორმატში (მაგ., teksti ან სურათი).

 **მოთხოვნა**: როდესაც თქვენ გადახვალთ ვებგვერდზე (მაგ., Google-ზე), თქვენი კომპიუტერი (აპლიკაციის ფენა) გზავნის მოთხოვნას სერვერთან, სადაც ეს ვებგვერდი ინახება.

 **პასუხი**: სერვერი იღებს თქვენს მოთხოვნას და აბრუნებს ინფორმაციას, რომელიც თქვენ გჭირდებათ (მაგ., ვებგვერდის კონტენტი), სწორ ფორმატში, რომელიც თქვენს ბრაუზერს ესმის.

 **მაგალითი**: თუ თქვენ YouTube-ზე ვიდეოს ეძებთ, აპლიკაციის ფენა ეხმარება თქვენს მოთხოვნას მიაღწიოს YouTube-ის სერვერს, და შემდეგ ვიდეოს ან სია გამოგიჩინოთ.

**სლაიდი 4**

**Client-Server არქიტექტურა**

ეს მოდელი არის ერთ-ერთი ყველაზე გავრცელებული და მარტივი პრინციპი, რომელსაც ინტერნეტში ძალიან ბევრი აპლიკაცია იყენებს.

**რა ხდება ამ მოდელში?**

* არსებობს **ერთი ძლიერი კომპიუტერი**, რომელსაც ეძახიან **სერვერს (Server)**.  
  ➤ ის **ყოველთვის ჩართულია** და მზად არის მოემსახუროს მოთხოვნებს.
* არიან **მომხმარებლები**, ანუ ჩვეულებრივი კომპიუტერები, ლეპტოპები ან ტელეფონები — მათ ეძახიან **კლიენტებს (Clients)**.  
  ➤ კლიენტები აგზავნიან მოთხოვნებს სერვერზე და სერვერი პასუხობს.

**მაგალითი:**

📄 წარმოიდგინე, რომ შენ ბრაუზერში (Chrome, Firefox) ეძებ www.example.com.

* შენი კომპიუტერი (კლიენტი) აგზავნის მოთხოვნას იმ ვებსერვერზე, სადაც example.com-ის საიტია გაშვებული.
* სერვერი უბრუნებს ვებსაიტის შინაარსს — სურათებს, ტექსტს და ყველაფერს.
* შენ მხოლოდ სერვერთან გაქვს კავშირი, სხვა მომხმარებლებთან არა.

### ****რა პრობლემა აქვს მხოლოდ ერთ სერვერზე დაყრდნობას?****

თუ ბევრი ადამიანი ერთდროულად იყენებს აპლიკაციას (მაგ. Facebook, Amazon), **ერთი სერვერი ვეღარ ასწრებს ყველას მომსახურებას**.

🔹 ამიტომ დიდ კომპანიებს (Google, Facebook, Amazon) აქვთ **დატა ცენტრები** – უზარმაზარი შენობები, სადაც ათასობით სერვერია მოთავსებული.

**სლაიდი 5**

Peer-to-Peer (P2P) არის მეთოდი, როდესაც კომპიუტერები (ან "ნოდები") ერთმანეთთან პირდაპირ უკავშირდებიან და ინფორმაციას უშუალოდ გადაეცემენ, სხვა ცენტრალური სერვერის გარეშე. ეს ჰგავს მეგობრების ჯგუფს, სადაც ყველა ერთმანეთთან თავად გადასცემს ნივთებს, ცალკე შუამავლის გარეშე.

**სლაიდი 6**

HTTP (Hypertext Transfer Protocol) არის წესი ან "ენა", რომელსაც ინტერნეტში გამოიყენებენ, რათა კომპიუტერი (კლიენტი) და სერვერი ერთმანეთთან ინფორმაცია გადასცენ. ის ჰგავს მომხმარებელსა და მაღაზიას შორის საუბარს: მომხმარებელი (კლიენტი) თხოვს რაღაცას, ხოლო მაღაზია (სერვერი) გადასცემს მას.ის შეიძლება იყოს **არამუდმივი** ან **მუდმივი**.

**სლაიდი 7**

**რა არის HTTP Request?** HTTP Request არის "მოთხოვნა", რომელსაც თქვენი კომპიუტერი (კლიენტი) უგზავნის სერვერს, როდესაც რამეს ითხოვთ ინტერნეტში, მაგალითად, სურათს ან გვერდს. ეს ჰგავს მაღაზიაში დარეკვას და თქმას: "მინდა ეს პროდუქტი!"

1. **Start Line (საწყისი ხაზი)**:
   * **Method (მეთოდი)**: ამბობს, თუ რა გინდა (აქ "GET" ნიშნავს, რომ სურათი ან ფაილი გინდა, მაგ., "background.png").
   * **Target (სამიზნე)**: ამბობს, რომელი ფაილი გჭირდება (მაგ., "/background.png").
   * **Version (ვერსია)**: ამბობს, რომელი HTTP ვერსია გამოიყენება (აქ "HTTP/1.0").
2. **Headers (ჩანართები)**:
   * იძლევა დამატებით ინფორმაციას, მაგ., "User-Agent: Mozilla/5.0" ნიშნავს, რომ მოთხოვნა ბრაუზერიდან მოდის.
3. **Body (სხეული)**:
   * "GET" მოთხოვნისთვის ხშირად ცარიელია (აქ "None"), რადგან მხოლოდ ფაილის მიღება გვინდა, არა მისი შეცვლა.

**სლაიდი 8**

**1. GET**

* **რა ნიშნავს?** GET ნიშნავს, რომ თქვენ უბრალოდ გინდათ რამეს "იხილოთ" ან "მიიღოთ" ინტერნეტიდან. ის ჰგავს მაღაზიაში შესვლას და თქმას: "გთხოვთ, მაჩვენოთ ეს ნივთი!"
* **როდის იყენებენ?** როდესაც გვერდს გადახვალთ (მაგ., Google-ზე) ან სურათს გადმოგიღებთ.
* **მაგალითი:** თუ თქვენ გადახვალთ [www.example.com/image.png](http://www.example.com/image.png), თქვენი ბრაუზერი სერვერს უთხრის: "მომაგებ background.png სურათი". სერვერი გიგზავნის მას, და თქვენ ხედავთ.
* **მახასიათებელი:** ეს მხოლოდ მიღებაა, არაფერს არ იცვლის ან არ აგზავნის.

**2. POST**

* **რა ნიშნავს?** POST ნიშნავს, რომ თქვენ გინდათ რამეს "გაგზავნოთ" ან "დაამატოთ" სერვერზე. ის ჰგავს მაღაზიაში წასვლას და თქმას: "გთხოვთ, აიღეთ ეს ნივთი და შეინახეთ!"
* **როდის იყენებენ?** როდესაც რაღაც ახალს ქმნით, მაგ., რეგისტრაციას გადიხართ ან ფორმას ავსებთ (მაგ., სახელის ან პაროლის შეტანა).
* **მაგალითი:** თუ ფორმაში ავსებთ თქვენს სახელს და ელფოსტას, POST მოთხოვნა გაგზავნის ამ ინფორმაციას სერვერზე, რომ ის შეინახოს.
* **მახასიათებელი:** აქ ჩვეულებრივ "Body" ნაწილია, სადაც ინფორმაცია გაიგზავნება.

**3. PUT**

* **რა ნიშნავს?** PUT ნიშნავს, რომ გინდათ რამეს "განაახლოთ" ან "შეცვალოთ" სერვერზე. ის ჰგავს მაღაზიაში დაბრუნებას და თქმას: "გთხოვთ, აიღეთ ეს ნივთი და შეცვალეთ ამით!"
* **როდის იყენებენ?** როდესაც არსებულ ინფორმაციას ცვლით, მაგ., პროფილის მონაცემების განახლება.
* **მაგალითი:** თუ თქვენს პროფილში სახელს იცვლით (მაგ., "გიორგი" → "გიო"), PUT მოთხოვნა გაგზავნის ახალ ინფორმაციას სერვერს, რომ ის განაახლდეს.
* **მახასიათებელი:** ეს ცვლილებას ახდენს, მაგრამ არ ქმნის ახალს.

**4. DELETE**

* **რა ნიშნავს?** DELETE ნიშნავს, რომ გინდათ რამეს "წაშალოთ" სერვერიდან. ის ჰგავს მაღაზიაში დარეკვას და თქმას: "გთხოვთ, წაშალოთ ეს ნივთი!"
* **როდის იყენებენ?** როდესაც რამეს შლით, მაგ., ანგარიშის წაშლა ან ფაილის წაშლა.
* **მაგალითი:** თუ თქვენ გადაწყვეტთ, რომ გაქვთ ზედმეტი ფაილი და გსურთ მისი წაშლა სერვერიდან, DELETE მოთხოვნა ამისთვის გამოიყენება.
* **მახასიათებელი:** ეს სამუდამოდ წაშლის ინფორმაციას.

**სლაიდი 9**

**\*\*რა არის HTTP Response?\*\***

HTTP Response არის "პასუხი", რომელსაც სერვერი გიგზავნის, როდესაც თქვენ ინტერნეტში რამეს ითხოვთ (მაგ., ვებგვერდი ან სურათი). ის ჰგავს მაღაზიის პასუხს, როდესაც თქვენ ნივთს ითხოვთ: "აქ არის, რაც გინდა, და ყველაფერი კარგადაა!"

**### როგორაა აგებული?**

1. \*\*HTTP Response Status Line (სტატუსის ხაზი)\*\*:

- ამბობს, თუ რა მდგომარეობაა. მაგ., "HTTP/1.1 200 OK" ნიშნავს, რომ ყველაფერი კარგადაა და თქვენი მოთხოვნა შესრულდა.

- "200 OK" ჰგავს თქმას: "გთხოვთ, აიღეთ ეს, ყველაფერი მზადაა!"

**2. \*\*HTTP Response Headers (ჩანართები)\*\*:**

- იძლევს დამატებით ინფორმაციას პასუხის შესახებ. მაგ.:

- \*\*Date:\*\* ამბობს, როდის გაიგზავნა პასუხი (აქ: ოთხ, 6 ივლისი 2022, 09:30:28).

- \*\*Accept-Ranges:\*\* ამბობს, რომ ფაილს ნაწილებადაც შეუძლია გაგზავნა (აქ: bytes).

- \*\*Content-Length:\*\* ამბობს, რამდენი ზომისაა ფაილი (აქ: 200 ბაიტი).

- \*\*Content-Type:\*\* ამბობს, რა ტიპის ინფორმაციაა (აქ: teksti UTF-8 ფორმატში).

**3. \*\*HTTP Response Body (სხეული)\*\*:**

- ამ ნაწილშია ის, რაც თქვენ ითხოვეთ. მაგ., თუ გვერდი ან სურათი გინდოდათ, ის აქ იქნება. აქ მაგალითად, ჩანს HTML კოდი (მაგ., "nav.navbar { ... some style }"), რაც ვებგვერდის დასაწყისს წარმოადგენს.

**სლაიდი 10**

**\*\*რა არიან HTTP Status Code-ები?\*\***

HTTP Status Code-ები არიან "სიგნალები", რომლებსაც სერვერი გიგზავნის, როდესაც თქვენ ინტერნეტში რამეს ითხოვთ. ისინი ჰგავს მაღაზიის პასუხს, რომელიც გიუთხრის, აი, რა მდგომარეობაა: "ყველაფერი კარგადაა", "უნდა დაელოდო" თუ "რაღაც არ მუშაობს". ეს კოდები იყოფა ჯგუფებად, რომლებიც სხვადასხვა მდგომარეობას ასახავს. განვიხილოთ თითოეული მარტივად!

**### 1xx (ინფორმაციული): "მოვლენაა, დაელოდე!"**

- ამბობს, რომ მოთხოვნა მიიღეს და პროცესი მიმდინარეობს. ჰგავს თქმას: "მხედველობაშია, ერთ წამში გავაგრძელებ!"

- მაგალითი: 100 - "დაელოდე, ვმუშაობ!"

**### 2xx (წარმატებული): "ყველაფერი კარგადაა!"**

- ნიშნავს, რომ თქვენი მოთხოვნა წარმატებით შესრულდა. ჰგავს: "აქ არის, რაც ითხოვე!"

- მაგალითი: 200 - "OK, გაგზავნილია!"

**### 3xx (გადამისამართება): "უნდა გადავიდე სხვაგან, დაიწყე!"**

- ამბობს, რომ თქვენი მოთხოვნა სხვა ადგილას გადავიდეს ან განახლდეს. ჰგავს: "ეს ახლა სხვა მაღაზიაშია, აი მისამართი!"

- მაგალითი: 301 - "გადადი ახალ მისამართზე."

**### 4xx (კლიენტის შეცდომა): "შენი ბრალია, შეამოწმე!"**

- ნიშნავს, რომ თქვენ რაღაც არასწორად გააკეთეთ. ჰგავს: "არასწორი მოთხოვნაა, გადაამოწმე!"

- მაგალითი: 404 - "გვერდი არ მოიძებნა."

**### 5xx (სერვერის შეცდომა): "ჩემი ბრალია, ვერ მოვასწარი!"**

- ამბობს, რომ სერვერს პრობლემა აქვს და ვერ შეძლო თქვენი მოთხოვნის შესრულება. ჰგავს: "ბოდიში, ჩემი შეცდომაა!"

- მაგალითი: 500 - "შიდა სერვერის შეცდომა."

**### მარტივი შედარება:**

- \*\*1xx\*\*: "დაელოდე!"

- \*\*2xx\*\*: "ყველაფერი მზადაა!"

- \*\*3xx\*\*: "გადადი ახალ ადგილზე!"

- \*\*4xx\*\*: "შენ გადაამოწმე!"

- \*\*5xx\*\*: "ჩემი პრობლემაა!"

**სლაიდი 11**

**### \*\*1. Without Parallel Connection (პარალელური კავშირის გარეშე):\*\***

- როცა ვებ-გვერდი იტვირთება, თითოეული ფაილი (HTML, სურათი, CSS) ცალ-ცალკე იგზავნება \*\*ერთი TCP კავშირით\*\*, ერთმანეთის მიყოლებით.

- მაგ.: თუ გვაქვს 3 ფაილი, ჯერ ერთი ჩაიტვირთება, მერე მეორე, მერე მესამე. \*\*ნელია\*\*, რადგან ყველაფერი რიგრიგობით ხდება.

**### \*\*2. With Parallel Connection (პარალელური კავშირებით):\*\***

- ერთდროულად \*\*რამდენიმე TCP კავშირი\*\* იხსნება, რათა სხვადასხვა ფაილი (HTML, სურათი, CSS) ერთდროულად ჩაიტვირთოს.

- მაგ.: 3 ფაილი ერთდროულად იტვირთება 3 ცალკე კავშირით. \*\*უფრო სწრაფია\*\*, მაგრამ მეტ რესურსს მოიხმარს.

**სლაიდი 12**

**### \*\*3. Pipelined (ფაიფლაინდი):\*\***

- ერთი TCP კავშირის გამოყენებით, მრავალი მოთხოვნა (request) იგზავნება \*\*ერთდროულად\*\*, სანამ სერვერის პასუხს დაველოდებით.

- მაგ.: 3 ფაილის მოთხოვნა ერთდროულად მიდის ერთ კავშირში, სერვერი ზედიზედ აგზავნის პასუხებს. \*\*სწრაფია\*\* და ნაკლებ რესურსს ხარჯავს.

**### \*\*4. Non-Pipelined (არაფაიფლაინდი):\*\***

- ერთ TCP კავშირში მოთხოვნები იგზავნება \*\*რიგრიგობით\*\*. ჯერ ერთი ფაილის მოთხოვნა მიდის, სერვერი პასუხობს, მერე მეორე ფაილის მოთხოვნა, და ა.შ.

- მაგ.: 3 ფაილისთვის 3 ცალკე მოთხოვნა-პასუხის ციკლი ხდება ერთ კავშირში. \*\*ნელია\*\*, რადგან ყოველ ჯერზე ელოდება პასუხს.

### \*\*შეჯამება:\*\*

- \*\*Without Parallel:\*\* ყველაფერი ნელა, ერთი კავშირით, რიგრიგობით.

- \*\*With Parallel:\*\* სწრაფი, მაგრამ მეტი კავშირი = მეტი რესურსი.

- \*\*Pipelined:\*\* სწრაფი, ერთი კავშირით, მოთხოვნები ერთდროულად.

- \*\*Non-Pipelined:\*\* ნელი, ერთი კავშირით, მოთხოვნები რიგრიგობით.

**სლაიდი 13**

**### \*\*რა არის Cookies?\*\***

Cookies (ქუქი-ფაილები) არის პატარა ტექსტური ფაილები, რომლებსაც ვებ-საიტები ინახავენ თქვენს კომპიუტერში, სმარტფონში ან სხვა მოწყობილობაში, როდესაც თქვენ სტუმრობთ მათ. ისინი შეიცავენ ინფორმაციას, რომელიც ეხმარება ვებ-საიტს „დაიმახსოვროს“ თქვენი ქმედებები, პრეფერენციები ან იდენტობა. Cookies ძირითადად გამოიყენება HTTP პროტოკოლის ფარგლებში, რადგან HTTP თავისთავად „მდგომარეობის გარეშეა“ (stateless), ანუ არ ინახავს ინფორმაციას მომხმარებლის წინა ქმედებების შესახებ.

**### \*\*Cookies-ის მიზანი\*\***

Cookies-ს რამდენიმე ძირითადი ფუნქცია აქვს:

1. \*\*მომხმარებლის იდენტიფიკაცია:\*\* ინახავს ინფორმაციას, როგორიცაა თქვენი სესიის ID, რათა საიტმა „გაიხსენოს“, რომ თქვენ უკვე შესული ხართ (მაგ., სოციალურ ქსელში).

2. \*\*პრეფერენციების შენახვა:\*\* ინახავს თქვენს არჩევანს (მაგ., ენის პარამეტრი, თემა: ღია/მუქი).

3. \*\*თვალყურის დევნება:\*\* გამოიყენება თქვენი ქცევის ანალიზისთვის, მაგ., რომელი გვერდები ნახეთ ან რა პროდუქტები დაათვალიერეთ.

4. \*\*რეკლამა:\*\* ეხმარება საიტებს, მოგაწოდოთ თქვენს ინტერესებზე მორგებული რეკლამები.

**### \*\*Cookies-ის ტიპები\*\***

**1. \*\*სესიის Cookies (Session Cookies):\*\***

- \*\*რა არის?\*\* დროებითი ქუქი-ფაილები, რომლებიც ინახება მხოლოდ მანამ, სანამ ბრაუზერს არ დახურავთ.

- \*\*მაგალითი:\*\* საიტზე შესვლისას თქვენი სესიის შენარჩუნება (რომ არ გადიოდეთ ხელახლა).

- \*\*თავისებურება:\*\* იშლება, როცა ბრაუზერს ხურავთ.

**2. \*\*მუდმივი Cookies (Persistent Cookies):\*\***

- \*\*რა არის?\*\* ინახება თქვენს მოწყობილობაში გარკვეული დროით (დღეები, თვეები, წლები).

- \*\*მაგალითი:\*\* „დამიმახსოვრე“ ფუნქცია, როცა საიტზე ავტომატურად შედიხართ.

- \*\*თავისებურება:\*\* აქვს ვადის გასვლის თარიღი

**3. \*\*მესამე მხარის Cookies (Third-Party Cookies):\*\***

- \*\*რა არის?\*\* სხვა დომენის მიერ შექმნილი ქუქი-ფაილები, ხშირად რეკლამისთვის.

- \*\*მაგალითი:\*\* Google Ads თვალყურს ადევნებს თქვენს ქცევას სხვადასხვა საიტზე.

- \*\*თავისებურება:\*\* ხშირად ბლოკავს ბრაუზერები კონფიდენციალურობის გამო.

**4. \*\*Zombie Cookies (ან Supercookies):\*\***

- \*\*რა არის?\*\* „აღდგება“ მაშინაც კი, თუ მომხმარებელი წაშლის მათ, რადგან ინახება სხვა ადგილებში (მაგ., ბრაუზერის ქეშში).

- \*\*მაგალითი:\*\* გამოიყენება თვალყურის დევნებისთვის, მაგრამ ითვლება არაეთიკურად.

**### \*\*როგორ მუშაობს Cookies?\*\***

**1. \*\*ქუქის შექმნა:\*\***

- როცა საიტს სტუმრობთ, სერვერი აგზავნის „Set-Cookie“ HTTP სათაურს, რომელიც ინახავს ქუქის თქვენს მოწყობილობაში.

- მაგ.: `Set-Cookie: user\_id=12345; Expires=Wed, 22 Jul 2026 11:27:00 GMT; Path=/`

**2. \*\*ქუქის გაგზავნა:\*\***

- ყოველ ჯერზე, როცა იმავე საიტს ეწვევით, თქვენი ბრაუზერი აგზავნის ქუქის სერვერზე „Cookie“ სათაურის მეშვეობით.

- მაგ.: `Cookie: user\_id=12345`

**3. \*\*სერვერის გამოყენება:\*\***

- სერვერი იყენებს ქუქის, რათა „გაიხსენოს“ თქვენი იდენტობა, პარამეტრები ან ქცევა.

**სლაიდი 14**

**### \*\*1. ვებ-ქეში (Web Cache)\*\***

- \*\*რა არის?\*\* ვებ-ქეში ინახავს ვებ-გვერდის ნაწილებს (მაგ., სურათები, HTML, CSS) თქვენს მოწყობილობაზე ან შუამავალ სერვერზე, რათა საიტი სწრაფად ჩაიტვირთოს მომავალში.

**\*\*როგორ მუშაობს?\*\***

- როცა საიტს ხსნით, ბრაუზერი ინახავს მონაცემებს (ქეშს).

- მეორედ საიტის გახსნისას ბრაუზერი იყენებს ქეშს, ნაცვლად იმისა, რომ ყველაფერი სერვერიდან გადმოწეროს.

- \*\*სარგებელი:\*\*

- \*\*სიჩქარე:\*\* ნაკლები დრო სჭირდება გვერდის ჩატვირთვას.

- \*\*ტრაფიკის შემცირება:\*\* ამცირებს სერვერის დატვირთვას.

- \*\*მაგალითი:\*\* ხშირად ნანახი საიტის ლოგო ან სურათი იტვირთება თქვენი ბრაუზერის ქეშიდან.

- \*\*უარყოფითი მხარე:\*\* თუ ქეში მოძველებულია, შეიძლება ძველი ვერსია ნახოთ.

**სლაიდი 15**

**Conditional HTTP GET**

- \*\*რა არის?\*\* ეს არის HTTP მოთხოვნა, რომელიც ამოწმებს, შეცვლილა თუ არა საიტის მონაცემები სერვერზე, სანამ ისევ გადმოწერს.

**\*\*როგორ მუშაობს?\*\***

- ბრაუზერი აგზავნის „If-Modified-Since“ ან „ETag“ სათაურს, რომელიც სერვერს ეკითხება: „შეცვლილა თუ არა ეს ფაილი?“

- თუ ფაილი არ შეცვლილა, სერვერი აბრუნებს „304 Not Modified“ და ბრაუზერი იყენებს ქეშს.

- თუ შეცვლილა, სერვერი გადმოგზავნის ახალ ფაილს.

- \*\*სარგებელი:\*\*

- ამცირებს მონაცემთა გადაცემას, თუ ფაილი უცვლელია.

- ზოგავს დროსა და ტრაფიკს.

- \*\*მაგალითი:\*\* თუ საიტის CSS ფაილი არ შეცვლილა, ბრაუზერი იყენებს ქეშს, ნაცვლად ხელახლა გადმოწერისა.

**სლაიდი 16**

SMTP (Simple Mail Transfer Protocol) არის „ენა“, რომელსაც ინტერნეტი იყენებს, რათა ელ.ფოსტა ერთი ადგილიდან მეორეზე გააგზავნოს. წარმოიდგინე, რომ ეს არის საფოსტო მანქანა, რომელიც წერილებს (ელ.ფოსტებს) აგზავნის სხვა ქალაქებში.

**როგორ მუშაობს SMTP?**

1. **წერილის დაწერა:**
   * თქვენ იყენებთ ელ.ფოსტის პროგრამას (მაგ., Gmail ან Outlook), დაწერეთ წერილი და დააჭირეთ „გაგზავნას“.
2. **SMTP-ის ჩართვა:**
   * თქვენი კომპიუტერი გადასცემს წერილს SMTP სერვერს (მაგ., Gmail-ის სერვერი). ეს სერვერი არის „საფოსტო ოფისი“, რომელმაც იცის, როგორ უნდა გააგზავნოს.
3. **გაგზავნა:**
   * SMTP სერვერი აგზავნის წერილს მიმღების სერვერზე (მაგ., თუ მეგობარს Gmail-ზე გაუგზავნით, მის Gmail სერვერს).
4. **მიღება:**
   * მიმღების სერვერი ინახავს წერილს, სანამ მეგობარი არ გახსნის მას (ამას POP3 ან IMAP ეხმარება).

**სლაიდი 17**

IMAP-ს თუ იყენებ, წერილები სერვერზე რჩება და შეგიძლია ნებისმიერი მოწყობილობიდან შეხვიდე და იგივე წერილები ნახო. ის ჰგავს Google Docs-ს – ყველაფერი ონლაინ არის და ყველგან ხელმისაწვდომია.

POP3 კი წერილებს ჩამოტვირთავს შენს კომპიუტერზე და სერვერიდან შლის. ამიტომ წერილები მხოლოდ იმ მოწყობილობაზე გაქვს, საიდანაც ჩამოიტვირთა. ეს ჰგავს Word ფაილს, რომელიც შენს კომპიუტერზე ინახება.

IMAP-ს მეტი სერვერის ადგილი სჭირდება და დაუცველია, თუ არ იყენებ დაშიფვრას (TLS). POP3 უსაფრთხოების მხრივ ცოტა მარტივია, მაგრამ მოქნილობა ნაკლებია.

**სლაიდი 18**

DNS (Domain Name System) არის ინტერნეტის "ტელეფონის საქაღალდე". ის გარდაქმნის ადვილად დასამახსოვრებელ სახელებს (მაგ., www.google.com) რთულ რიცხვებად (IP მისამართებად, მაგ., 142.250.190.14), რომლებიც კომპიუტერებს ესმით

**რას აკეთებს?**

**1. \*\*სახელის IP-ად გარდაქმნა (hostname-to-IP-address translation):\*\***

- \*\*რა არის?\*\* ამბობს, თუ რა IP მისამართია ვებ-გვერდის სახელი.

- \*\*მაგალითი:\*\* როცა www.facebook.com-ს ხსნი, DNS ამბობს: „ეს არის 157.240.241.35“. შენი კომპიუტერი ამ მისამართზე მიდის, როგორც მეგობრის სახლში.

**2. \*\*სახელის გადარქმევა (host aliasing):\*\***

- \*\*რა არის?\*\* იძლევს სხვა სახელებს ერთი საიტისთვის.

- \*\*მაგალითი:\*\* www.shop.com-ს შეიძლება ჰქონდეს მეორე სახელი, როგორც „store.shop.com“. DNS ამბობს, რომ ორივე ერთია.

- \*\*canonical, alias names:\*\* ერთი ძირითადი სახელი (canonical, მაგ., www.shop.com) და მეორე სახელები (alias, მაგ., store.shop.com).

**3. \*\*მეილის დახმარება (mail server aliasing):\*\***

- \*\*რა არის?\*\* ეხმარება ელ.ფოსტებს, რომ იცნონ ერთმანეთი სხვა სახელებით.

- \*\*მაგალითი:\*\* თუ მეილს mail.google.com-ს გაუგზავნი, DNS იცის, რომ ეს იგივეა, რაც gmail.com.

**4. \*\*ტვირთის გაყოფა (load distribution):\*\***

- \*\*რა არის?\*\* ნიშნავს, რომ ბევრი ადამიანი ერთდროულად გამოიყენებს საიტს, და DNS ხალხს განაწილებს.

- \*\*მაგალითი:\*\* თუ 1000 ადამიანი youtube.com-ს ხსნის, DNS ამბობს: „100 ადამიანი ამ სერვერზე, 100 სხვაზე“, რათა ყველა სწრაფად ჩაიტვირთოს.

**5. \*\*ბევრი მისამართი ერთი სახელისთვის (replicated Web servers):\*\***

- \*\*რა არის?\*\* ერთი სახელს (მაგ., google.com) შეიძლება ჰქონდეს რამდენიმე IP.

- \*\*მაგალითი:\*\* google.com-ს შეიძლება ჰქონდეს 3 IP (1.1.1.1, 2.2.2.2, 3.3.3.3), და DNS არჩევს უახლოესს.

**სლაიდი 19**

## როგორ ხდება, როცა ვწერთ www.amazon.com?

შენი კომპიუტერი (DNS client) უნდა გაიგოს, რა **IP მისამართზე** არის ეს საიტი.

მაგრამ მას IP არ ახსოვს და ეკითხება DNS სერვერებს, თანმიმდევრობით ასე:

### 1.****კითხვა Root DNS სერვერს****

→ "გთხოვ, მითხარი ვინ იცის .com საიტების მისამართები?"

* Root DNS სერვერებმა იციან, სად არის .com, .org, .edu და სხვების სერვერები.
* ისინი პასუხობენ: "აი, მიდი .com DNS სერვერთან".

**მაგალითი:** თითქოს ქუჩაში კითხავ ვინ იცის "კომპანიის" მისამართი, და გეუბნებიან – „მიდი რეგისტრაციის ოფისში, ისინი იციან“.

### 2️.****კითხვა .com DNS სერვერს****

→ "გთხოვ, მითხარი amazon.com-ს ვინ მართავს?"

* .com სერვერი პასუხობს: "აი, amazon.com-ის DNS სერვერის მისამართი".

**მაგალითი:** რეგისტრაციის ოფისში გეუბნებიან – „amazon-ის ოფისი აი აქ არის“.

### 3️.****კითხვა amazon.com DNS სერვერს****

→ "მითხარი, რა IP აქვს [www.amazon.com-ს](http://www.amazon.xn--com--eer/)?"

* ახლა უკვე მივდივართ იმ ორგანიზაციის სერვერთან, ვისაც მართლა ეს საიტი ეკუთვნის.
* ის პასუხობს: "[www.amazon.com](http://www.amazon.com/) არის მაგალითად → 192.0.2.1".

**მაგალითი:** ბოლოს amazon-ის ოფისში გეუბნებიან – „ჩვენი ვებსაიტი აქაა, აი მისი მისამართი“.

**სლაიდი 20**

## **Local DNS Name Servers** – ადგილობრივი DNS სერვერები

### როგორ მუშაობს?

* როცა კომპიუტერი (ჰოსტი) აგზავნის DNS მოთხოვნას, ის იგზავნება მის **ლოკალურ DNS სერვერზე**
* ეს ლოკალური სერვერი ან:
  + პასუხობს თვითონ (თუ ქეში აქვს),
  + ან აგზავნის მოთხოვნას DNS-ის გლობალურ იერარქიაში (Root → TLD → Authoritative)

### სად ინახება ინფორმაცია?

* ლოკალურ DNS სერვერს აქვს **ქეში** — ბოლოს გამოყენებული სახელებისა და მისამართების ჩამონათვალი
* ქეში შეიძლება იყოს **ძველი/მოძველებული**, ამიტომ ყოველთვის არაა 100% სანდო

### ვინ გვაძლევს ლოკალურ DNS სერვერს?

* ყოველი ინტერნეტ მომწოდებელი (ISP) გთავაზობს თავის **DNS სერვერს**
  + მაგალითად: Silknet, Magti, Comcast და სხვ.

**Recursive Query – "ყველაფერი შენ მოაგვარე!"**

* კლიენტი (მაგ. შენი კომპიუტერი) ეუბნება **ლოკალურ DNS სერვერს**:  
  **"მინდა IP მისამართი www.amazon.com–ის. შენ მოძებნე და დამიბრუნე საბოლოო პასუხი!"**
* ლოკალური DNS სერვერი იწყებს მთელი DNS იერარქიის გადაკითხვას:  
  Root → TLD (.com) → amazon.com → იღებს საბოლოო პასუხს
* **ლოკალური DNS სერვერი პასუხობს კლიენტს უკვე დასრულებული შედეგით**

**მაგალითი:** შენ ეკითხები მეგობარს – „გთხოვ, იპოვე ამ მაღაზიის მისამართი და მეც მომიტანე პასუხი!“ და ის **თავისი ძალებით** ყველგან ეძებს.

**Iterative Query – "მითხარი ვის ვკითხო!"**

* კლიენტი ან DNS სერვერი ეკითხება სხვა DNS სერვერს:  
  **"იცი www.amazon.com-ის IP?"**
* თუ პასუხი არ იცის, პასუხობს:  
  **"მე არ ვიცი, მაგრამ აი ვის უნდა ჰკითხო – წადი ამ სერვერთან"**
* კლიენტი **თვითონ აგრძელებს** შემდეგ სერვერთან მოთხოვნას

**მაგალითი:** შენ ეკითხები ვიღაცას – „სად არის ეს მაღაზია?“ და ის გეუბნება – „მე არ ვიცი, მაგრამ ნახე იმ ოფისში!“ შენ მიდიხარ და ისევ ეკითხები სხვას.

**DNS Caching – ქეშირება DNS-ში**

* როდესაც **ნებისმიერი DNS სერვერი** ერთხელ "ისწავლის" დომენის მისამართს (მაგ. www.amazon.com → 192.0.2.1),  
  ის **შეინახავს** ამ ინფორმაციას თავის ქეშში (მეხსიერებაში)
* შემდეგი მსგავსი მოთხოვნისას, **მოაქვს პასუხი ქეშიდან** – სწრაფად, ზედმეტი ძებნის გარეშე
* **Caching აუმჯობესებს რეაგირების სისწრაფეს** – არ ხდება ყოველ ჯერზე Root/TLD სერვერებზე კითხვა

**სლაიდი 21**

**DHCP (Dynamic Host Configuration Protocol)** არის სისტემა, რომელიც ავტომატურად გასცემს თქვენს კომპიუტერს ინტერნეტში მუშაობისთვის საჭირო ინფორმაციას. ის ჰგავს მაღაზიის მმართველს, რომელიც გასცემს თქვენთვის შესაბამის ნომერს და მისამართს, რომ შეძლოთ შესვლა!

**\*\*რას გასცემს DHCP?\*\***

1. \*\*IP Addresses (IP მისამართები)\*\*: ეს არის თქვენი კომპიუტერის "უნიკალური ნომერი" ინტერნეტში, მაგ., 192.168.1.10.

2. \*\*Subnet Mask (ქვე-ქსელის ნიღაბი)\*\*: ეხმარება გაიგოს, თუ რომელ ნაწილშია თქვენი ქსელი.

3. \*\*Default Gateway (ნაგულისხმევი კარიბჭე)\*\*: ეს არის "მთავარი კარიბჭე", რომლითაც ინტერნეტში გადადიხართ.

4. \*\*DNS Server (DNS სერვერი)\*\*: ეხმარება იპოვნოს ვებგვერდები, მაგ., www.google.com.

**სლაიდი 22**

### 1. DISCOVER (ძებნა)

* **რა ხდება?** კლიენტი (თქვენი კომპიუტერი) ეძებს DHCP სერვერს, რომელიც მისცემს ინფორმაციას.
* **მაგალითი:** ჰგავს მაღაზიაში შესვლას და თქმას: "ვინ მომცემს ბარათს?"
* **სურათში:** კლიენტი "DISCOVER" მესიჯს უგზავნის.

### 2. OFFER (შეთავაზება)

* **რა ხდება?** DHCP სერვერი პასუხობს და სთავაზობს კლიენტს IP მისამართს.
* **მაგალითი:** მაღაზიის მმართველი ამბობს: "აქაა შენი ბარათი, გამოიყენე ეს ნომერი!"
* **სურათში:** სერვერი "OFFER" გაგზავნის.

### 3. REQUEST (მოთხოვნა)

* **რა ხდება?** კლიენტი აირჩევს შეთავაზებულ IP მისამართს და ითხოვს მის დადასტურებას.
* **მაგალითი:** თქვენ ამბობთ: "დიახ, მსურს ეს ნომერი, გთხოვთ, დაადასტურეთ!"
* **სურათში:** კლიენტი "REQUEST" უგზავნის.

### 4. ACK (დადასტურება)

* **რა ხდება?** DHCP სერვერი დადასტურებს IP მისამართს და გადასცემს მას კლიენტს, რათა ინტერნეტში გამოიყენოს.
* **მაგალითი:** მმართველი ამბობს: "OK, ნომერი შენია, გამოიყენე ინტერნეტი!"
* **სურათში:** სერვერი "ACK" გაგზავნის.

### მარტივად:

* **DISCOVER:** "ვეძებ სერვერს!"
* **OFFER:** "აქაა შენი IP!"
* **REQUEST:** "მინდა ეს IP!"
* **ACK:** "OK, გამოიყენე!"

**სლაიდი 23**

**ძირითადი ფუნქციები:**

* ასრულებს მონაცემების გარდაქმნას ისეთ ფორმატში, რომელიც მარტივად აღსაქმელია აპლიკაციის ფენისთვის
* უზრუნველყოფს მონაცემების დაშიფვრასა და გაშიფვრას გადაცემისა და მიღების პროცესში
* ახორციელებს მონაცემების შეკუმშვას, რათა ისინი უფრო სწრაფად და ეფექტიანად გადაეცეს სესიის ფენას

**Presentation ფენა – ფუნქციები**  
წარდგენა წარმოადგენს OSI მოდელის მე-6 ფენას და ასრულებს მნიშვნელოვან როლს, რათა აპლიკაციის ფენამ მიიღოს სწორად დამუშავებული და მისთვის გასაგები ინფორმაცია. ამ ფენას აქვს რამდენიმე ძირითადი ამოცანა:

1. **მონაცემების წარმოდგენის გარდაქმნა**
   * ეს თარგმნა არის **Presentation Layer-ში მონაცემების გარდაქმნა** – ანუ ინფორმაცია გადადის იმ ენაზე/ფორმატზე, რასაც მიმღები გაიგებს. მაგალითად,
   * თუ აგზავნი ტექსტს, ის გადაიქცევა **Unicode/ASCII** ფორმატში. თუ აგზავნი სურათს, ის გარდაიქმნება **JPEG/PNG** ფორმატში.
2. **დაშიფვრა და გაშიფვრა (Encryption / Decryption)**
   * როდესაც მონაცემები იგზავნება, წარდგენის ფენა ახდენს მათ დაშიფვრას უსაფრთხოების მიზნით.
   * მიღებისას კი იმავე ფენაზე ხდება გაშიფვრა, რათა მომხმარებელმა მიიღოს მონაცემები წაკითხვისთვის ხელმისაწვდომი ფორმით.
3. **შეკუმშვა და გაშლაც (Compression / Decompression)**
   * მონაცემების გაგზავნამდე, ფენა მათ ამცირებს (შეკუმშავს), რათა ტრანსმისია მოხდეს უფრო სწრაფად და ეფექტურად.
   * მიღებისას მონაცემები კვლავ იშლება თავდაპირველ ზომამდე, რათა ზუსტად იქნას წარდგენილი.
4. **სინტაქსის და სემანტიკის უზრუნველყოფა**
   * ფენა უზრუნველყოფს, რომ ორივე მხარეს გამოყენებული მონაცემების სტრუქტურა და მნიშვნელობა იყოს თავსებადი, ანუ მხარეები ერთმანეთს სწორად გაუგონ.

**სლაიდი 24**

## **რა არის SSL/TLS?**

**SSL (Secure Sockets Layer)** და მისი გაუმჯობესებული ვერსია **TLS (Transport Layer Security)** — არის **დაცული საკომუნიკაციო პროტოკოლები**, რომლებიც გამოიყენება ინტერნეტზე ინფორმაციის დაშიფრული გადაცემისთვის.

თანამედროვე ვებსაიტებზე ვიყენებთ მხოლოდ **TLS-ს** (მაგ. HTTPS://), SSL პრაქტიკულად აღარ გამოიყენება, მაგრამ ტერმინი „SSL“ ჯერ კიდევ ხშირად გვხვდება.

## **რისთვის გვჭირდება SSL/TLS?**

* დაშიფვრა — მონაცემები ვერ წაიკითხება გზაში (მაგ. პაროლები, ბარათის ნომრები).
* ავთენტიკაცია — გვარწმუნებს, რომ სწორ სერვერთან ვმუშაობთ.
* მონაცემების მთლიანობა — გზაში არავის შეუძლია მონაცემების შეცვლა.

ამ სურათში ნაჩვენებია როგორ მუშაობს **TLS (Transport Layer Security)** პროტოკოლი კავშირის დასაყენებლად **კლიენტსა და სერვერს** შორის.

## **TLS კავშირის დამყარება**

**TLS არის პროტოკოლი**, რომელიც იცავს შენს პირადობას და ინფორმაციას როცა იყენებ ინტერნეტს (მაგალითად ბრაუზერში შედიხარ ვებსაიტზე HTTPS-ით).

### 1. ****ClientHello****

* **კლიენტი (მაგ. შენი ბრაუზერი)** ამბობს: "გამარჯობა სერვერო, მინდა დაგიკავშირდე უსაფრთხოდ".
* ამ შეტყობინებაში ის უგზავნის სერვერს:
  + რომელი **TLS ვერსიით** შეუძლია მუშაობა
  + რომელ **შიფრებს** (ალგორითმებს) იყენებს დაშიფვრისთვის
  + აგზავნის „random“ რიცხვს (შემთხვევით რიცხვს), რომელიც საჭიროა დაშიფვრისთვის.

### 2. ****ServerHello + სერტიფიკატი****

* **სერვერი პასუხობს** კლიენტს:
  + ირჩევს დაშიფვრის ალგორითმს (ერთ-ერთს იმათგან რაც კლიენტმა შესთავაზა)
  + აგზავნის თავის **სერტიფიკატს** (რომელშიც არის საჯარო გასაღები და მტკიცებულება რომ ნამდვილი სერვერია)
  + აგზავნის „random“ რიცხვს თავის მხარესაც
  + ამბობს: "მზად ვარ უსაფრთხო კავშირის დასაწყებად".

### 3. ****ClientKeyExchange****

* კლიენტი იღებს სერვერის სერტიფიკატს და საჯარო გასაღებით ქმნის **საიდუმლო გასაღებს** (shared secret).
* ეს საიდუმლო გასაღები მხოლოდ კლიენტმა და სერვერმა იციან და ის გამოიყენება შიფრაციისთვის.
* კლიენტი აგზავნის:
  + ამ საიდუმლო გასაღებს, დაშიფრულად.
  + ChangeCipherSpec: „ახლა უკვე ვიწყებთ დაშიფრულ კომუნიკაციას“
  + Finished: ბოლო შეტყობინება ამ ეტაპზე, უკვე დაშიფრულია.

### 4. ****ChangeCipherSpec + Finished (სერვერიდან)****

* სერვერი პასუხობს:
  + ChangeCipherSpec: „ჰო, მეც ვიწყებ დაშიფრულად საუბარს“
  + Finished: ბოლოჯერ ამოწმებს, რომ ყველაფერი სწორად მოხდა.

**სლაიდი 25**

**Session Layer** არის OSI მოდელის **მეხუთე ფენა** (Layer 5), რომელიც მართავს "სესიის" ანუ **კომუნიკაციის აქტიურ ურთიერთობას ორ მოწყობილობას შორის**.

ანუ როდესაც ერთი აპლიკაცია მეორეს უკავშირდება (მაგ. Zoom ზარი, ვიდეო სტრიმინგი, ან ფაილის გადაცემა), ეს ფენა უზრუნველყოფს, რომ:

* კავშირი სწორად და წესრიგიანად დაიწყოს
* მთელი კომუნიკაციის განმავლობაში ყველაფერი სინქრონში იმუშაოს
* და როცა ყველაფერი დასრულდება — კავშირი სწორად დაიხუროს

## მთავარი ფუნქციები:

### 1. ****სესიის დამყარება (Session Establishment)****

ეს ფენა აყალიბებს "სესიას" ანუ დროებით კომუნიკაციას ორ მხარეს შორის, როგორიცაა:

* კლიენტი ↔ სერვერი
* მომხმარებელი ↔ აპლიკაცია

მაგალითად: როცა Zoom-ზე უკავშირდები სხვის კამერას, session layer აშენებს ამ დიალოგს.

### 2. ****დიალოგის კონტროლი (Dialog Control)****

ეს ფენა წყვეტს — **ვინ როდის ლაპარაკობს**.

* თუ სისტემები მუშაობენ **half-duplex-ით** (ერთი ლაპარაკობს, მეორე უსმენს) — session layer აქ იყენებს ტოკენებს ("ჩემი ჯერია!")
* თუ **full-duplex-ით** მუშაობენ (ორივე ლაპარაკობს ერთდროულად) — ის ამასაც უჭერს მხარს

### 3. ****სინქრონიზაცია (Synchronization)****

სესიის ფენა მონაცემების ნაკადში რთავს სპეციალურ **სინქრონიზაციის წერტილებს (checkpoints)**. ეს ნიშნავს:

* თუ კავშირი გაწყდება, შენ არ დაკარგავ მთელ კომუნიკაციას
* შეგიძლიათ გააგრძელოთ იქიდან, სადაც გაჩერდა

მაგალითად: ვიდეოს ყურებისას თუ ინტერნეტი გაწყდა, როცა დაუბრუნდები, სისტემამ იცის ზუსტად საიდან განაახლოს.

### 4. ****აქტივობების მენეჯმენტი (Activity Management)****

დატა იყოფა დამოუკიდებელ ნაწილებად (activities), რაც კავშირს მოქნილს და სწორად სამართავს ხდის.

### 5. ****სესიის დახურვა (Session Termination)****

როცა კომუნიკაცია სრულდება, session layer უზრუნველყოფს **დაცულად დახურვას**, ისე რომ არაფერი დაიკარგოს.

## როგორ მუშაობს Session Layer?

1. კლიენტი აგზავნის მოთხოვნას → სერვერთან სესიის დასაწყებად
2. ხდება ავთენტიკაცია (შესაძლოა პაროლით, სერტიფიკატით და ა.შ.)
3. იდგმება წესები: როგორ ილაპარაკებენ (token-based თუ simultaneous)
4. დაიდება სინქრონიზაციის წერტილები
5. გადაეცემა მონაცემები
6. დაიხურება სესია

**სლაიდი 26**

**სლაიდი 27-28**

### ****Multiplexing**** — გაგზავნა:

***"მრავალი აპლიკაციის მონაცემის ერთ არხზე გადაგზავნა"***

როცა შენს კომპიუტერზე რამდენიმე აპლიკაცია ინტერნეტს იყენებს (მაგალითად, ბრაუზერი, Spotify და Zoom), ყველა მათგანი მონაცემებს აგზავნის ერთნაირად — **IP მისამართისა და Transport Layer პროტოკოლის (TCP/UDP) მეშვეობით.**

მაგრამ როგორ გაიგებს ქსელი, რომელი მონაცემი ეკუთვნის რომელ აპლიკაციას?

აქ ჩნდება Multiplexing:

* Transport layer სხვადასხვა აპლიკაციებისთვის სხვადასხვა **port ნომერს** იყენებს.
* მაგალითად:
  + ბრაუზერი იყენებს 443 პორტს (HTTPS)
  + ელ. ფოსტა იყენებს 25 პორტს (SMTP)
  + Zoom შესაძლოა გამოიყენოს 8801 პორტი და ა.შ.

Transport Layer ყველა აპლიკაციის მონაცემს აერთიანებს ერთ „გზავნად“ და გზავნის IP-ზე TCP/UDP headers-ით.

### ****Demultiplexing**** — მიღება:

***"მიღებული მონაცემების გაყოფა სწორ აპლიკაციებზე"***

როცა სერვერი ან სხვა მოწყობილობა იღებს მონაცემებს, transport layer უნდა მიხვდეს:

* რომელი პაკეტი რომელ აპლიკაციას ეკუთვნის?
* ეს ხდება **port ნომრების გამოყენებით** (source port, destination port).

**Demultiplexing** გაშიფრავს მონაცემებს და სწორად გადააწვდიდა კონკრეტულ აპლიკაციას.

## 🔢 მაგალითი

### შენს კომპიუტერზე:

* YouTube → ბრაუზერი → Port 443
* Spotify → App → Port 4070

**Multiplexing**: ბრაუზერის და Spotify-ის მონაცემები ერთდროულად გადის ქსელში სხვადასხვა პორტებით.

### სერვერზე:

**Demultiplexing**: მიღებული პაკეტები თავიანთი პორტის მიხედვით იგზავნება შესაბამის სერვისზე.

**სლაიდი 29**

**Checksum** არის რიცხვი, რომელიც გამოიყენება იმის შესამოწმებლად, გაფუჭდა თუ არა მონაცემები გადაცემისას.

ის არის:

* პატარა რიცხვი (მაგ. 16 ბიტი),
* გამომუშავებული მონაცემებისგან,
* იძლევა საშუალებას მიმღებმა შეამოწმოს, სწორად მივიდა თუ არა მონაცემი.

## **როგორ მუშაობს Checksum?**

### 1. ****გაგზავნის მხარე (Sender):****

* იღებს მთელ მონაცემებს (data),
* იყენებს სპეციალურ ალგორითმს (მაგ. one's complement sum),
* ითვლის Checksum-ს,
* გზავნის მონაცემებთან ერთად.

### 2. ****მიმღები მხარე (Receiver):****

* იღებს მონაცემებს და Checksum-ს,
* თავიდან ითვლის Checksum-ს მიღებული მონაცემებიდან,
* ადარებს მიღებულ Checksum-ს გამოთვლილს:
  + თუ ემთხვევა ✅ → მონაცემები არ დაზიანებულა.
  + თუ არ ემთხვევა ❌ → მონაცემები დაზიანდა გზაში.

**სლაიდი 30**

**სლაიდი 31**

**Transmission Control Protocol (TCP)** — ეს არის კავშირზე ორიენტირებული კომუნიკაციის პროტოკოლი, რომელიც სხვადასხვა მოწყობილობას შორის ქსელზე შეტყობინებების გადაცვლაში ეხმარება. ის TCP/IP პროტოკოლთა ერთ-ერთი მთავარი ნაწილია. **OSI მოდელში**, TCP მუშაობს **ტრანსპორტის ფენაში (Layer 4)**. ის მდებარეობს აპლიკაციისა და ქსელის ფენებს შორის და უზრუნველყოფს მონაცემების საიმედო მიწოდებას. TCP თანამშრომლობს **IP პროტოკოლთან**, რომელიც განსაზღვრავს, თუ როგორ იგზავნება მონაცემთა პაკეტები კომპიუტერებს შორის.

**როგორ მუშაობს TCP?**

TCP ყოფს მონაცემებს მცირე ნაწილებად (პაკეტებად) და შემდეგ მეორე ბოლოზე აწყობს ისევ მთლიანად, რათა დარწმუნდეს, რომ შეტყობინება სრულად და დაზიანების გარეშე მიაღწია დანიშნულების ადგილს. მონაცემების მცირე ნაწილებად გაგზავნა უფრო ეფექტურია, ვიდრე ერთიანად მთელი ინფორმაციის გაგზავნა.

ამ პატარა პაკეტებს შეუძლიათ იმოძრაონ სხვადასხვა მარშრუტით, თუ რომელიმე გზა გადატვირთულია, თუმცა საბოლოო დანიშნულება მაინც იგივეა.

**ტრანსპორტის ფენის (Transport Layer – Layer 4) ფუნქციები:**

**სერვისები, რომლებსაც აპლიკაციებისთვის შეიძლება TCP ან UDP უზრუნველყოფდეს (ან არა):**

* **საიმედო მონაცემთა გადაცემა** – დარწმუნება, რომ ყველაფერი მივიდა სრულად და სწორად
* **შეცდომის აღმოფხვრა (Error Recovery)** – დაზიანებული ან დაკარგული მონაცემების თავიდან გაგზავნა
* **მონაცემთა თანმიმდევრობა (Data Sequencing)** – ინფორმაციის სწორად დალაგება მიღებისას
* **ნაკადების კონტროლი (Flow Control)** – გადაცემის სიჩქარის მორგება მიმღების შესაძლებლობაზე

**TCP არის "კავშირზე ორიენტირებული" პროტოკოლი**  
→ სანამ ერთ კომპიუტერი მეორესთან მონაცემებს გაუგზავნის, ისინი პირველ რიგში ერთმანეთთან "კავშირს ამყარებენ" – ანუ ელაპარაკებიან ერთმანეთს და ეთანხმებიან კომუნიკაციის დაწყებაზე.  
→ კავშირი რომ ერთხელ დადგინდა, მხოლოდ მაშინ იწყება მონაცემების გაცვლა.

**TCP უზრუნველყოფს საიმედო კომუნიკაციას**  
→ როცა ერთი კომპიუტერი მეორეს მონაცემს უგზავნის, მეორე უნდა დაადასტუროს, რომ მიიღო ("Acknowledgment" – მიღების დასტური).  
→ თუ მიღების დასტური არ მოვიდა, იგივე მონაცემი თავიდან იგზავნება.

**TCP უზრუნველყოფს სწორ თანმიმდევრობას (Sequencing)**  
→ TCP-ში არსებობს რიგის ნომრები, რომლებიც ეხმარება მიმღებს, დაალაგოს ნაწილებად გამოგზავნილი მონაცემები სწორად – თუნდაც თუ ისინი არასწორ თანმიმდევრობით მივიდნენ.

**TCP უზრუნველყოფს ნაკადის კონტროლს (Flow Control)**  
→ მიმღები კომპიუტერი ეუბნება გამგზავნს, რომ დააჩქაროს ან შეანელოს გაგზავნა, რათა არ გადატვირთოს თავისი სისტემა.

შეგიძლია ასე დაიმახსოვრო:  
TCP = დაინტერესებული მეგობარი, რომელიც ჯერ გკითხავს მზად ხარ თუ არა საუბრისთვის (კავშირი), დაგელოდება რომ ყველაფერი გაიგო (საიმედოობა), გეტყვის რა და როდის თქვა (თანმიმდევრობა), და გთხოვს არ იჩქარო, თუ ბევრი ინფორმაცია გაქვს (ნაკადის კონტროლი).

**TCP უზრუნველყოფს ამ ყველაფერს**, ხოლო **UDP — არა** (მარტივი, სწრაფია, მაგრამ ნაკლებად სანდო)

**სლაიდი 32**

ეს სურათი აჩვენებს **TCP Segment Header**-ს სტრუქტურას – ანუ როგორ გამოიყურება ინფორმაცია, როცა გადადის ქსელში TCP პროტოკოლით. TCP მონაცემებს ანაწილებს პატარა ნაწილებად (segments), თითოეულს თავში ერთვის ეს „ჰედერი“, სადაც წერია ყველა აუცილებელი ინფორმაცია, რომ მონაცემი სწორად მივიდეს და დადასტურდეს. ახლა მარტივად აგიხსნი თითოეულ ველს:

**1. Source Port Number (2 bytes)**

ეს არის **გამომგზავნის პორტი**.  
მაგალითად, თუ შენი კომპიუტერი აგზავნის მონაცემებს ვებსერვერზე, ის იყენებს შემთხვევით გამოყოფილ პორტს (მაგ. 49500), რომ სერვერმა იცოდეს, საიდან მოვიდა პაკეტი.

**2. Destination Port Number (2 bytes)**

ეს არის **მიმღების პორტი**.  
მაგალითად, ვებსერვერი ისმენს 80 პორტზე (HTTP) ან 443-ზე (HTTPS). ასე იცის TCP-მ, რომელი აპლიკაციისთვის არის პაკეტი განკუთვნილი.

**3. Sequence Number (4 bytes)**

ეს არის **მონაცემების რიგითობა**.  
TCP ანაწილებს დიდ მონაცემს პატარა ნაწილებად. Sequence Number აჩვენებს, ეს კონკრეტული ნაწილი რომელია ჯაჭვში, რომ მიმღებმა სწორად შეაერთოს ყველა ნაწილი თავიდან ბოლომდე.

**4. Acknowledgement Number (4 bytes)**

ეს არის **დადასტურების ნომერი**.  
მიმღები ამ ნომრით უგზავნის პასუხს გამგზავნს: „მე მივიღე მონაცემები ამ ნომრამდე, შეგიძლია შემდეგი გამომიგზავნო“. ასე TCP უზრუნველყოფს სანდო მიწოდებას.

**5. Data Offset (4 bits)**

აჩვენებს, **საიდან იწყება ნამდვილი მონაცემი** ჰედერის შემდეგ, რადგან ჰედერის სიგრძე შეიძლება ცვალებადი იყოს (არის optional data).

**6. Reserved (3 bits)**

ეს ბიტები ჯერ არაფერში გამოიყენება, უბრალოდ რეზერვშია მომავალი ფუნქციებისთვის.

**7. Control Flags (9 bits)**

ეს არის პატარა ნიშნები, რომლებიც აჩვენებს, რას აკეთებს პაკეტი. მაგალითად:

* **SYN** – კავშირის დასაწყებად
* **ACK** – მიღების დასადასტურებლად
* **FIN** – კავშირის დასასრულებლად
* **RST, PSH, URG** – სხვა სპეციალური ფუნქციებისთვის

**8. Window Size (2 bytes)**

ამით მიმღები ეუბნება გამგზავნს, **რამდენი ბაიტი შეუძლია მიიღოს ერთდროულად**, რომ არ გადატვირთოს ბუფერი. ეს ეხმარება TCP-ს ნაკადის მართვაში (Flow Control).

**9. Checksum (2 bytes)**

ეს არის **შეცდომების შემოწმება**.  
გამგზავნი ითვლის ჰედერისა და მონაცემების checksum-ს და აყენებს აქ. მიმღები ამოწმებს იგივე წესით. თუ განსხვავდება, პაკეტი დაზიანებულია და ხელახლა ითხოვს.

**10. Urgent Pointer (2 bytes)**

თუ მონაცემი სასწრაფოა (URG flag ჩართულია), ეს ველი აჩვენებს, რომელამდეა სასწრაფო ინფორმაცია. ჩვეულებრივ იშვიათად გამოიყენება.

**11. Optional Data (0–40 bytes)**

ეს არის დამატებითი ინფორმაცია, მაგალითად: **Maximum Segment Size (MSS)**, **Window Scaling**, ან სხვა გაფართოებები, რომლებიც აუმჯობესებს TCP-ის მუშაობას.

მოკლედ, TCP ჰედერი არის ბარათი, რომელიც პაკეტთან ერთად მიდის და ეუბნება მიმღებს:

* ვინაა გამგზავნი და მიმღები (პორტები)
* რომელი ნაწილია ეს (Sequence Number)
* მიღებულია თუ არა წინა ნაწილები (Acknowledgement Number)
* რამდენი შეიძლება ერთდროულად მიიღო (Window Size)
* და სწორია თუ არა პაკეტი (Checksum).

ეს ყველაფერი ერთად უზრუნველყოფს, რომ TCP იყოს **სანდო, თანმიმდევრული და კონტროლირებადი კავშირი**.

თუ გინდა, შემიძლია შემდეგში **გაგიმარტივო პატარა მაგალითით**, როგორ მუშაობს ეს პრაქტიკაში. გინდა?

**სლაიდი 33-34**

პორტები აუცილებელია, რომ განსაზღვრო **რომელი აპლიკაცია** მიიღებს მონაცემებს.

მაგალითად:

* ვებსერვერი იყენებს პორტს **80** (HTTP)
* FTP იყენებს პორტს **21**
* და ასე შემდეგ...

**პორტების დიაპაზონები (IANA-ს მიერ განსაზღვრული):**

| **პორტის ტიპი** | **დიაპაზონი** | **აღწერა** |
| --- | --- | --- |
| **Well-known (ცნობილი)** | 0 – 1023 | სისტემის ძირითადი სერვისები (HTTP, FTP, DNS...) |
| **Registered (რეგისტრირებული)** | 1024 – 49151 | პროგრამებისთვის, რომლებიც დარეგისტრირებულია IANA-ში |
| **Ephemeral/Private/Dynamic** | 49152 – 65535 | დროებითი პორტები კლიენტის მხარეს |

**სლაიდი 35**

**Tcp threeway handshake**

| **ეტაპი** | **ვინ აგზავნის** | **რას აგზავნის** | **რისთვის** |
| --- | --- | --- | --- |
| 1 | კლიენტი | SYN | კავშირის დაწყება |
| 2 | სერვერი | SYN-ACK | თანხმობა და საკუთარი SYN |
| 3 | კლიენტი | ACK | თანხმობის დადასტურება |

**შედეგი:**  
ამ სამი ნაბიჯის შემდეგ ორივე მხარეს აქვს ინფორმაცია ერთმანეთზე, იციან როდის და როგორ გადააგზავნონ მონაცემები და იწყება სანდო კომუნიკაცია TCP-ს საშუალებით.

**სლაიდი 36**

### TCP Four-Way Handshake

**როდესაც ერთი მხარე (კლიენტი ან სერვერი) უნდა დაასრულოს კავშირი TCP-ში, გამოიყენება ოთხნაბიჯიანი პროცესი, რომელსაც ვეძახით "four-way handshake".**

ეს აუცილებელია იმისთვის, რომ მონაცემების გადაცემა უსაფრთხოდ დასრულდეს და არ დაიკარგოს ბოლო პაკეტები.

### როგორ მუშაობს Four-Way Handshake?

#### წარმოდგინე: კლიენტი და სერვერი უკვე ურთიერთობენ (კავშირი უკვე დამყარებულია — ანუ 3-way handshake უკვე მოხდა და ისინი "ლაპარაკობენ").

### ◼️ ნაბიჯი 1 — ****FIN (Finish):****

კლიენტი (ან სერვერი) აგზავნის FIN პაკეტს:  
„მე დავასრულე მონაცემების გაგზავნა, მზად ვარ კავშირი გავწყვიტო.“

➡ ეს ნიშნავს, რომ ის აღარ გააგზავნის მონაცემებს, მაგრამ შეუძლია კიდევ მიიღოს მეორე მხარისგან.

### ◼️ ნაბიჯი 2 — ****ACK (Acknowledgment):****

მეორე მხარე პასუხობს ACK:  
„შენი მოთხოვნა მივიღე, მესმის რომ გინდა გაწყვეტა.“

➡ ამ ეტაპზე კავშირი ჯერ კიდევ ღიაა მეორე მხარის მხრიდან.

### ◼️ ნაბიჯი 3 — ****FIN (Finish):****

ახლა მეორე მხარეც აგზავნის FIN პაკეტს:  
„მეც დავასრულე გაგზავნა, მზად ვარ გაწყვეტისთვის.“

### ◼️ ნაბიჯი 4 — ****ACK (Final Acknowledgment):****

დაბოლოს, პირველი მხარე პასუხობს ACK პაკეტით:  
„კარგი, შენი დასრულებაც მივიღე — კავშირი სრულად გაწყვეტილია.“

### შედეგი:

ამის შემდეგ ორივე მხარე კავშირს წყვეტს და აღარ ხდება არც გაგზავნა და არც მიღება.

**სლაიდი 37**

**TCP (Transmission Control Protocol)** არის გადაცემის პროტოკოლი, რომელიც უზრუნველყოფს:

✅ მონაცემების სწორად და თანმიმდევრულად გაგზავნას  
✅ დაკარგული ნაწილის თავიდან გაგზავნას  
✅ მიღებულის დადასტურებას – რომ მივიდა რაც უნდა მისულიყო

ამაში **Sequencing (თანმიმდევრობა)** და **Acknowledgment (დადასტურება)** ძალიან დიდ როლს თამაშობენ.

## წარმოიდგინე სიტუაცია:

შენ ხარ გამგზავნი – A  
მეგობარი მიმღები – B  
და შენ მას 100 გვერდიანი წიგნი უნდა გადაუგზავნო… თუმცა ერთიანად ვერ უგზავნი – პატარა ნაწილებად ტეხავ.

### როგორ დარწმუნდები, რომ:

* ყველა გვერდი მივიდა?
* არცერთი არ გამოტოვებულა?
* არცერთი არ გაორმაგდა?
* სწორ რიგში მიდის?

ამისთვის გამოიყენება **Sequencing** და **Acknowledgment**.

## 1️.Sequencing – „თანმიმდევრობის ნომერი“

TCP ყოველ ნაწილს (segment-ს) აძლევს უნიკალურ ნომერს, რომელსაც ეწოდება **Sequence Number**.მაგალითად:

* პირველი ნაწილი – Sequence Number = 1
* მეორე ნაწილი – Sequence Number = 101 (ანუ – პირველი 100 ბაიტიანი იყო)
* მესამე – Sequence Number = 201  
  და ასე შემდეგ…

ამით მიმღები იცის, რომ **რა იყო რიგით პირველი, მეორე, მესამე...** – და შეუძლია სწორად დაალაგოს ყველაფერი.

## 2️.Acknowledgment – „დადასტურება, რომ მივიდა“

როცა მიმღები მიიღებს რომელიმე ნაწილს, ის აბრუნებს გაგზავნილს შეტყობინებით:  
**"Acknowledgment Number = N"** – რაც ნიშნავს:  
**„შემდეგს ველოდები ნომრით N“**

მაგალითად:

* გაგზავნილია 100 ბაიტიანი სეგმენტი, Sequence Number = 1
* მიმღები აბრუნებს – Acknowledgment Number = 101  
   ანუ ეს ნიშნავს: **"პირველი ნაწილი მივიღე, მომწერე 101-დან აგრძელე"**

## და თუ რამე არ მივიდა?

* TCP არ მიიღებს Acknowledgment-ს → ხვდება რომ ნაწილი არ მივიდა → **თავიდან აგზავნის**.

მაგალითად:

* გაგზავნე Sequence 1–100, მაგრამ 101–200 არ მივიდა.
* არ მიიღებ Acknowledgment = 201
* TCP იტყვის: “მიდი თავიდან 101-200!"

**სლაიდი 38**

## **რა არის TCP Retransmission?**

**Retransmission** ნიშნავს — **გადაგზავნილ მონაცემების ხელახლა გაგზავნას**, როცა არ მივიღეთ დადასტურება (ACK), ან მივიდა დაზიანებული სეგმენტი.

### რატომ ხდება ხელახლა გაგზავნა?

1. **სეგმენტი დაიკარგა** გზაში
2. **სეგმენტი დაზიანდა** (გავიგეთ Checksum-ით)
3. **ACK არ მივიღეთ დროულად** (timeout)

### როგორ ამჩნევს TCP რომ ხელახლა უნდა გაგზავნოს?

#### TCP იყენებს ორ მექანიზმს:

### 1. ****Timeout-based Retransmission****

* გაგზავნის შემდეგ TCP იწყებს ტაიმერს.
* თუ **ACK არ მოვიდა დროულად**, სეგმენტი **ხელახლა იგზავნება**.

მაგალითი:

* Sequence = 1000 გაგზავნილია
* Timeout 1 წამია
* თუ 1 წელში ACK = 2000 არ მოვიდა → TCP ხელახლა აგზავნის sequence = 1000

### 2. ****Fast Retransmit**** (არ დაელოდოს timeout-ს)

* თუ მივიღეთ **3 ერთი და იგივე ACK** (duplicate ACKs), TCP ეჭვობს რომ რაღაც დაკარგულია.

მაგალითი:

* მივიდა ACK 1000, მერე ისევ ACK 1000, ისევ ACK 1000
* TCP ფიქრობს, რომ რაღაც რიგში არ არის  
  → **მაშინვე აგზავნის** დაკარგულს, ტაიმერის ლოდინის გარეშე

**სლაიდი 39**

## რა არის Flow Control?

**Flow Control** ნიშნავს **გამგზავნმა რამდენი მონაცემი უნდა გაგზავნოს ისე, რომ მიმღები არ გადაიტვირთოს.**

წარმოგიდგინე რომ:

* შენ აგზავნი წერილებს
* შენი მეგობარი კი კითხულობს

მაგრამ მეგობარს წაკითხვა რომ აყოვნდება და შენ ზედმეტად ბევრს უგზავნი — შეიძლება ზოგი წერილი საერთოდ დაეკარგოს ან დაიკარგოს რიგში.

ამიტომ TCP იყენებს **Flow Control-ს** — რათა გამგზავნმა იცოდეს **"რამდენამდე შეიძლება გაგზავნო, რომ მიმღები არ გადაიღალოს."**

## და რა შუაშია Window Size?

### Window Size ნიშნავს:

**რამდენ ბაიტს შეუძლია მიმღებს ერთდროულად მიიღოს Acknowledgment-ის გარეშე.**

## მარტივი მაგალითი:

წარმოიდგინე TCP მიმღები ამბობს:

„მე მაქვს 300 ბაიტიანი ფანჯარა (Window Size = 300) – ანუ შეგიძლია გამომიგზავნო ერთბაშად 300 ბაიტი, და მერე დაიცადე ჩემი დადასტურებამდე.“

* გამგზავნი აგზავნის პირველ 300 ბაიტს
* ელის Acknowledgment-ს
* როგორც კი მოვიდა Acknowledgment = 301
* გაგზავნის შემდეგ 300-ს

## რატომ გვჭირდება ეს?

თუ მიმღები გადატვირთულია – შეუძლია თქვას:

„Window Size = 0“ – ანუ „არ გამომიგზავნო ჯერ, ვერ ვასწრებ დამუშავებას.“

ან შეიძლება თქვას:

„Window Size = 1000“ – „წესრიგში ვარ, შეგიძლია ბევრიც გამომიგზავნო.“

## ტექნიკურად:

* **Window Size** არის **TCP Header-ში** არსებული ველი – 16 ბიტიანი
* მაქსიმუმი = 65,535 ბაიტი (თუ არ იყენებს Window Scaling-ს)

**სლაიდი 40-41**

რა არის UDP? (User Datagram Protocol)

UDP არის ძალიან მარტივი პროტოკოლი, რომელიც გამოიყენება **მონაცემების გაგზავნისთვის ინტერნეტში**.

ის განსხვავდება TCP-სგან, რადგან:

**UDP = სწრაფი, მაგრამ არაჯერიანი გაგზავნა.**

UDP-ის ძირითადი თვისებები :

### 1️.****არ ქმნის კავშირს (Not connection-oriented)****

UDP არ ამოწმებს, მზად არის თუ არა მიმღები.  
→ უბრალოდ აგზავნის, ვინმეს თუ დაუვარდა – ეგ უკვე მისი პრობლემაა

**მაგალითად:**  
როგორც პარკში მყოფი ადამიანი, რომელიც ყვირის: "ვისაც უნდა, მოისმინოს!"

### 2️.****არ არის სანდო (No reliability)****

→ არ აგზავნის **დადასტურებას** (Acknowledgment).  
→ თუ მონაცემი დაიკარგა გზაში, **თავიდან არ იგზავნება.**

### 3️.****არ აწყობს თანმიმდევრობას (No sequencing)****

→ არ იყენებს Sequence Number-ებს.  
→ თუ ნაწილები არასწორად მივიდა (1-3-2), UDP ვერ მიხვდება.

### 4️.****არ აქვს ნაკადების კონტროლი (No flow control)****

→ TCP-ს მსგავსი window size არ აქვს.  
→ ვერ აკონტროლებს „რამდენი“ მონაცემი უნდა გაგზავნო.

## სად ვიყენებთ UDP-ს?

რადგან UDP სწრაფია და „არ ჭკუობს“, ვიყენებთ მაშინ როცა:

* **სისწრაფე უფრო მნიშვნელოვანია, ვიდრე სანდოობა:**

ვიდეოჩატი, ონლაინ თამაშები, live აუდიო  
→ უკეთესია რაღაც პატარა ინფორმაცია დაიკარგოს, ვიდრე მთლიანად შეაყოვნოს ყველაფერი.

**სლაიდი 42**

**ქსელის ფენის ძირითადი ამოცანებია:**

* **ლოგიკური მისამართების მინიჭება**
* **მარშრუტიზაცია**
* **ინკაპსულაცია**
* **დაშლა და ხელახლა აწყობა**
* **შეცდომების დამუშავება**

**სლაიდი 43**

ეს სურათი არის **IPv4 პაკეტის ჰედერის სტრუქტურა**, ანუ როგორია ინტერნეტში გაგზავნილი პაკეტის თავი, სადაც ჩაწერილია ყველა მნიშვნელოვანი ინფორმაცია, რომ პაკეტი სწორად მივიდეს დანიშნულების ადგილას. ახლა მარტივად და ვრცლად აგიხსნი თითო ველი რას ნიშნავს:

**1. Version (4 bits)**

ეს აჩვენებს **IP-ის ვერსიას**. აქ ყოველთვის 4 წერია, რადგან IPv4 პაკეტია.

**2. IHL – Internet Header Length (4 bits)**

ეს არის **ჰედერის სიგრძე** 32-ბიტიან სიტყვებში. სტანდარტულად 20 ბაიტია, მაგრამ თუ დამატებითი პარამეტრები (Options) არსებობს, სიგრძე იზრდება.

**3. DSCP და ECN (8 bits)**

* **DSCP (Differentiated Services Code Point)** – Quality of Service (QoS)-ისთვის გამოიყენება, რომ ზოგიერთი პაკეტი უფრო პრიორიტეტული იყოს (მაგ. ვიდეოზარი).
* **ECN (Explicit Congestion Notification)** – სიგნალი, რომ ქსელში გადატვირთვაა, და გამგზავნმა უნდა შეანელოს სიჩქარე.

**4. Total Length (16 bits)**

ეს არის **პაკეტის მთლიან სიგრძე ბაიტებში**, ჰედერისა და მონაცემების ჩათვლით. მაქსიმუმი 65,535 ბაიტია.

**5. Identification (16 bits)**

თუ დიდი პაკეტი დაიშალა რამდენიმე ნაწილად (Fragment), თითოეულ ნაწილს ეს ნომერი მიენიჭება, რომ მიმღებმა იცოდეს, რომელი ფრაგმენტები ერთად უნდა შეიკრას.

**6. Flags (3 bits)**

ეს პატარა ნიშნებია, რომლებიც აკონტროლებს **ფრაგმენტაციას**.  
მაგალითად:

* 0 – შეიძლება გაიფრაგმენტოს
* 1 – არ გაიფრაგმენტოს
* 2 – ეს ბოლო ფრაგმენტია თუ არა

**7. Fragment Offset (13 bits)**

ეს აჩვენებს, **რა პოზიციაზე დგას ფრაგმენტი მთლიან პაკეტში**, რომ მიმღებმა სწორად შეაერთოს.

**8. Time to Live – TTL (8 bits)**

ეს არის პაკეტის სიცოცხლის დრო ჰოპებში. ყოველი როუტერი TTL-ს აკლებს ერთით. როცა ნულს მიაღწევს, პაკეტი იშლება, რომ უსასრულოდ არ იაროს ქსელში.

**9. Protocol (8 bits)**

აჩვენებს, პაკეტის შიგნით რომელი პროტოკოლის მონაცემებია.  
მაგალითად:

* 6 → TCP
* 17 → UDP
* 1 → ICMP

**10. Header Checksum (16 bits)**

ეს არის **ჰედერის შეცდომების შემოწმება**. თუ ჰედერი დაზიანდა გზაში, მიმღები ამას ამჩნევს და პაკეტს უარყოფს.

**11. Source IP Address (32 bits)**

ეს არის **გამომგზავნის IP მისამართი**. მაგალითად: 192.168.1.10

**12. Destination IP Address (32 bits)**

ეს არის **მიმღების IP მისამართი**, სადაც უნდა მივიდეს პაკეტი.

**13. Options (თუ IHL > 5)**

ეს არის დამატებითი პარამეტრები, რომლებიც იშვიათად გამოიყენება, მაგალითად უსაფრთხოებისთვის ან ტესტირებისთვის.

**მოკლედ**:  
IPv4 ჰედერი არის პაკეტის პასპორტი.  
ის ეუბნება ქსელს:

* ვინ აგზავნის (Source IP)
* ვისთან მიდის (Destination IP)
* რამდენი უნდა იცოცხლოს (TTL)
* რა პროტოკოლია შიგნით (TCP, UDP…)
* დაშლილია თუ არა ნაწილებად (Fragment + Offset)
* და სწორია თუ არა პაკეტი (Checksum).

ამით ინტერნეტი სწორად ახერხებს პაკეტების მიტანას საბოლოო მისამართზე.

გინდა, შემდეგში მოგიხსნა **სუპერ მარტივი ყოველდღიური მაგალითით**, რომ ამ ჰედერს ადვილად დაიმახსოვრებ?

**სლაიდი 44**

 **რა არის?** IPv4 მისამართი შედგება ოთხი რიცხვისგან, რომლებიც წერტილებითაა გparted (მაგ., 192.168.1.1). ყოველი რიცხვი 0-დან 255-მდეა, და ეს ჯამში ქმნის 32-ბიტიან კოდს.

 **რატომ მნიშვნელოვანია?** ეს მისამართი საშუალებას აძლევს კომპიუტერებს ერთმანეთს ინტერნეტში "იპოვნონ".

 **რა პრობლემაა?** IPv4-ს მხოლოდ დაახლოებით 4.3 მილიარდი უნიკალური მისამართი აქვს (2³²), რაც დღეს უკვე ცოტაა, რადგან ინტერნეტში მოწყობილობების რაოდენობა (ტელეფონები, კომპიუტერები, IoT მოწყობილობები) გაიზარდა.

 **რატომ მნიშვნელოვანია?** ამის გამო გამოჩნდა IPv6, მაგრამ IPv4 ჯერ კიდევ გამოიყენება.

 **IP მისამართის ნაწილები:**

* **ქსელის ნაწილი**: განსაზღვრავს, თუ რომელ ქსელშია კომპიუტერი (მაგ., 192.168).
* **ჰოსტის ნაწილი**: განსაზღვრავს კონკრეტულ მოწყობილობას ქსელში (მაგ., 1.1).

 **Subnet Mask**: ეხმარება გაარჩიოს ქსელი და ჰოსტი (მაგ., 255.255.255.0).

**სლაიდი 45**

## **ლოგიკური მისამართება (Logical Addressing)**

### ****რა არის ლოგიკური მისამართი?****

ლოგიკური მისამართი — ეს არის უნიკალური IP მისამართი (მაგ: 192.168.1.1), რომელიც კომპიუტერს ან სხვა მოწყობილობას ენიჭება ქსელში, რათა შეძლოს კომუნიკაცია სხვა მოწყობილობებთან.

იგი გამოიყენება იმისთვის, რომ მონაცემები სწორად მივიდეს გამგზავნიდან მიმღებამდე **გლობალურ ან ადგილობრივ ქსელში**.

### ****როგორ ენიჭება ლოგიკური მისამართი?****

ლოგიკური მისამართი შეიძლება მიენიჭოს **ორივე გზით**:

1. **სტატიკურად (Static IP Addressing):**
   * ხელით ენიჭება მოწყობილობას
   * გამოიყენება იმ მოწყობილობებზე, რომლებიც მუდმივად ერთ IP მისამართზე უნდა იყვნენ (მაგ: სერვერები)
2. **დინამიკურად (Dynamic IP Addressing):**
   * ავტომატურად ენიჭება **DHCP სერვერის** მეშვეობით
   * როგორც წესი, გამოიყენება ჩვეულებრივ მომხმარებელ მოწყობილობებზე (მაგ: ტელეფონები, ლეპტოპები Wi-Fi-ზე)

**სლაიდი 46**

**სლაიდი 47**

**სლაიდი 48**

**რა ხდება ინტერნეტში მონაცემების გაგზავნისას?** როდესაც თქვენ ინტერნეტში რამეს გაგზავნით (მაგ., ელფოსტა ან ვიდეო), თქვენი კომპიუტერი (გამგზავნი) და მიმღები (მაგ., სერვერი) იყენებენ სპეციალურ პროცესებს, რომლებიც ჰგავს "ჩანთის შეფუთვას" და "ჩანთის გახსნას". ეს ორი პროცესი ყველაზე მნიშვნელოვანია: **ენკაფსულაცია** (Encapsulation) და **დეკაფსულაცია** (Decapsulation). განვიხილოთ მარტივად!

**1. ენკაფსულაცია (Encapsulation) - "ჩანთის შეფუთვა"**

* **რა არის?** ეს არის პროცესი, როდესაც გამგზავნი (Sender) იღებს თქვენს მონაცემებს (მაგ., teksti ან ფაილი) და "ჩასვამს" მას სხვადასხვა ფენის "ჩანთებში" (Headers), რათა ის უსაფრთხოდ გადაეცეს.
* **როგორ ხდება?**
  1. **აპლიკაციის ფენა**: თქვენ იწყებთ მონაცემებით (მაგ., "გამარჯობა!").
  2. **პრეზენტაციისა და სესიის ფენები**: მონაცემები "ფორმატირდება" (მაგ., teksti-ს კოდად გარდაქმნა).
  3. **ტრანსპორტის ფენა**: უერთვის L4 Header (მაგ., ზომა და თანმიმდევრობა).
  4. **ქსელის ფენა**: ამატებს L3 Header (მაგ., IP მისამართი).
  5. **მონაცემთა ბმის ფენა**: ამატებს L2 Header და Trailer (მაგ., ქსელის მისამართი).
  6. **ფიზიკური ფენა**: აკონვერტირებს მონაცემები ბიტებად (0 და 1), რაც ხაზით გადადის.

**რატომ მნიშვნელოვანია ეს?**

* **ენკაფსულაცია** უზრუნველყოფს, რომ მონაცემები სწორად "ჩასვას" და გზაზე არ დაიკარგოს.
* **დეკაფსულაცია** იძლევა საშუალებას, რომ მიმღებმა გაიგოს, რა გააგზავნა.
* ეს ორი პროცესი ერთად ჰგავს წერილის გაგზავნასა და მიღებას: ჩასმა, გაგზავნა, გახსნა და წაკითხვა!

**სლაიდი 49**

## რა არის Fragmentation?

**Fragmentation** ნიშნავს მონაცემის გაყოფას პატარა ნაწილებად.  
ეს ხდება მაშინ, როდესაც:

ერთი დიდი IP პაკეტი **ვერ ეტევა** ქსელის ფიზიკურ დათმობილობაზე (მაგ. Ethernet-ის ფრეიმში), ანუ ის უფრო დიდია, ვიდრე მაქსიმალური დასაშვები ზომა – **MTU (Maximum Transmission Unit)**.

## სად და რატომ ხდება Fragmentation?

Fragmentation **IP ფენაში** (Layer 3) ხდება.

მაგალითად, თუ აპლიკაციამ გააგზავნა 4000 ბაიტის სიგრძის პაკეტი, მაგრამ რომელიღაც ქსელში მაქსიმალური დასაშვები MTU არის 1500 ბაიტი, მაშინ IP ფენა აიღებს იმ 4000 ბაიტიან პაკეტს და **გააყოფს სამ ან მეტ ნაწილად** (ფრაგმენტად), რომლებიც შეიძლება გადაეცეს ქსელში.

## 🔹 Fragmentation-ის პროცესის ძირითადი ცნებები:

| **ტერმინი** | **განმარტება** |
| --- | --- |
| **MTU** | მაქსიმალური ბაიტების რაოდენობა, რომელსაც ქსელი ან ლინკი იტევს ერთ პაკეტში. |
| **Fragment** | გაყოფილი IP პაკეტის ერთ-ერთი ნაწილი. |
| **Offset** | აჩვენებს პაკეტის რომელ ნაწილთან არის ეს ფრაგმენტი დაკავშირებული. |
| **More Fragments (MF) Flag** | 1 ნიშნავს: კიდევაა ფრაგმენტები; 0 ნიშნავს: ეს ბოლო ფრაგმენტია. |
| **Identification** | უნიკალური ნომერი, რომლითაც მიმღები იცნობს რომელი ფრაგმენტები ეკუთვნის რომელ პაკეტს. |

## Reassembly (ხელახლა აწყობა)

ეს პროცესია, როცა **მიმღები მოწყობილობა** იღებს ფრაგმენტებს და აწყობს პირვანდელ IP პაკეტად.

Reassembly **მხოლოდ დანიშნულების მოწყობილობაზე** ხდება – არანაირი შუალედური როუტერი არ აკეთებს ამ პროცესს.

📦 მოწყობილობა იყენებს:

* **Identification** ველს → რომელი ფრაგმენტები ეკუთვნის ერთ მთლიან პაკეტს.
* **Offset** ველს → რომ დაალაგოს ფრაგმენტები სწორ თანმიმდევრობაში.
* **MF Flag** → რომ გაიგოს, ფრაგმენტები სრულად მივიდა თუ არა.

**სლაიდი 50-51**

**Data Link Layer** არის OSI მოდელის **მეორე ფენა**, რომელიც მდებარეობს ფიზიკურ ფენასა (Layer 1) და ქსელის ფენას (Layer 3) შორის.

მისი მთავარი ამოცანებია:

* მონაცემების გადაცემა ერთი მოწყობილობიდან მეორეზე **მოწესრიგებულად და შეცდომების გარეშე**
* ფიზიკურ კაბელზე ან უკაბელო არხზე უსაფრთხო და ორგანიზებული კომუნიკაცია
* IP პაკეტის ჩასმა Frame-ში და ამ Frame-ის გადაცემა MAC მისამართების გამოყენებით

### 1. ****Framing (ფრემის შექმნა)****

ფრემინგი ნიშნავს იმ პროცესს, როცა ქსელური პაკეტი იყოფა პატარა ერთეულებად – **Frame-ებად**, რომლებსაც დამატებული აქვთ:

* **Header** – ინფორმაციის დასაწყისი (მიმღებისა და გამომგზავნის MAC მისამართები)
* **Payload** – შიგნით მოთავსებული მონაცემი (მაგ., IP პაკეტი)
* **Trailer** – მონაცემის ბოლოს მიმაგრებული FCS (Frame Check Sequence), რომელიც გამოიყენება შეცდომის დასადგენად

### 2. ****Flow Control (ნაკადის კონტროლი)****

ეს ფუნქცია უზრუნველყოფს, რომ:

* გამგზავნმა არ „გადააჭარბოს“ მონაცემების გაგზავნას
* მიმღები შეძლებს მიღებული მონაცემების დამუშავებას

მაგალითად: თუ მიმღები მოწყობილობა გადატვირთულია, ის აცნობებს გამომგზავნს, რომ შეანელოს ტემპი.

### 3. ****Error Detection and Correction (შეცდომის აღმოჩენა და გამოსწორება)****

ფიზიკური ფენა არ იძლევა შეცდომებისგან დაცვას, ამიტომ Data Link Layer იყენებს **FCS-ს**:

* გამგზავნი ქმნის სპეციალურ კოდს, რომელიც ეფუძნება გაგზავნილ ბიტებს
* მიმღები ახდენს იგივე კოდის გამოთვლას მიღებული ბიტების საფუძველზე
* თუ არ ემთხვევა – ფრემი უარყოფილია ან ითხოვს ხელახალ გადაგზავნას

### 4. ****Reliable Data Delivery (სანდო მიწოდება)****

ამ ფენაში ვხედავთ **დადასტურებას**, რომ მონაცემები მიღებულია:

* Receiver გზავნის Acknowledgement-ს
* თუ არ მიიღო, გამგზავნი აგზავნის თავიდან

### 5. ****Media Access Control (MAC – მედიის წვდომის კონტროლი)****

MAC ქვეფენა აგვარებს, თუ რომელი მოწყობილობა გააგზავნის მონაცემებს:

* **Collision avoidance** – Wi-Fi-ზე თუ ორი მოწყობილობა ერთდროულად გააგზავნის, მონაცემები იკარგება
* **Half Duplex** – ერთდროულად ან მიღება ან გაგზავნა
* **Full Duplex** – ერთდროულად გაგზავნა/მიღება

**სლაიდი 52**

**სლაიდი 53**

LLC არის ქსელის ფენა, რომელიც მოქმედებს, როგორც ხიდი ზედა ფენებს (მაგ., Network Layer) და ქვედა ფენას (Physical Layer) შორის, რათა ისინი ერთმანეთთან კარგად ილაპარაკონ. ის პასუხისმგებელია იმაზე, რომ იდენტიფიციროს, თუ რომელი პროტოკოლი (მაგ., IPv4 ან IPv6) იყენებს მონაცემებს, და აკონტროლებს მონაცემთა ნაკადს, რომ არ მოხდეს გადატვირთვა. მისი მთავარი დავალებაა, რომ შეამოწმოს შეცდომები გადაცემის დროს და შეატყობინოს ზედა ფენებს, თუ რამე გაუჭირდა. მაგალითად, როდესაც შენ აგზავნი IP პაკეტს, LLC ეუბნება MAC ფენას, თუ რა ტიპის მონაცემია ეს (მაგ., IPv6) და მზადყოფნას უკარნახებს მის გასაგზავნად.

MAC არის ფენა, რომელიც მუშაობს ძალიან დაბალ დონეზე და ზრუნავს იმაზე, თუ როგორ გამოიყენება ფიზიკური მედიუმი, მაგ., კაბელი ან Wi-Fi. ის განსაზღვრავს, თუ რომელმა მოწყობილობამ და როდის უნდა გამოიყენოს ეს მედიუმი, რომ არ მოხდეს "შეტაკება". მისი მთავარი ფუნქციაა, რომ აიღოს მონაცემები LLC-დან, ჩასვას ის Frame-ში, სადაც ხდება Destination და Source MAC მისამართების დამატება, ასევე შეცდომების შემოწმების ველი (FCS). მაგალითად, როდესაც შენი კომპიუტერი მონაცემებს აგზავნის, MAC ქმნის Frame-ს, ამოწმებს, თუ კაბელი თავისუფალია, და მხოლოდ მერე გაგზავნის ინფორმაციას სწორ მისამართზე.

**სლაიდი 54**

## როგორ მუშაობს **CSMA/CD**? (გამოიყენება კაბელზე)

### ეტაპები:

1. **Carrier Sense (შემოწმება)**  
   მოწყობილობა უსმენს კაბელს — არის თუ არა თავისუფალი?
2. **Multiple Access (რამდენიმე წვდომა)**  
   ყველა მოწყობილობას შეუძლია გადაცემა, მაგრამ მხოლოდ მაშინ, როცა არხი თავისუფალია.
3. **Collision Detection (შეჯახების აღმოჩენა)**  
   თუ შემთხვევით ორ მოწყობილობას ეგონა რომ არხი თავისუფალია და ერთდროულად გააგზავნეს ➤ **შეჯახება მოხდა**.
4. **გადაწყვეტა**
   * შეწყვეტენ გადაცემას
   * აგზავნიან **jam სიგნალსს**, რომ სხვებმაც გაიგონ
   * ელოდებიან შემთხვევით დროს (**backoff**) და ხელახლა ცდილობენ

**სლაიდი 55**

Physical Layer არის ინტერნეტის ყველაზე დაბალი ფენა, რომელიც ზრუნავს იმაზე, თუ როგორ გადაეცემა ინფორმაცია ფიზიკურად ერთი ადგილიდან მეორეზე. ის ჰგავს საფოსტო სერვისს, სადაც წერილი ფიზიკურად უნდა გადაეცეს მიმღებთან. ამ ფენას სჭირდება ნამდვილი ნივთები, მაგ., კაბელები, Wi-Fi ტალღები ან სხვა მოწყობილობები, რომ მონაცემები გადაეცეს. სურათზე ნაჩვენებია საფოსტო ყუთი, მანქანა, თვითმფრინავი და მუშა, რაც ასახავს, თუ როგორ "მგზავრობს" ინფორმაცია.

ეს ფენა იღებს მონაცემებს ბიტების სახით (0 და 1) და გადასცემს მათ კაბელის ან ტალღების მეშვეობით. მაგალითად, როდესაც შენ Wi-Fi-ს იყენებ, Physical Layer ცვლის ინფორმაციას ტალღებად, რომ ის ჰაერით გადავიდეს. ის ასევე ზრუნავს, რომ კაბელები ან მოწყობილობები სწორად იყოს დაკავშირებული, რომ არ მოხდეს შეცდომა. ამრიგად, ეს ფენა ჰგავს გზას, რომელიც ინფორმაციას მიმღებთან აერთებს!

**სლაიდი 56**

**Fiber Optic Cable (ოპტიკური კაბელი):** ოპტიკური კაბელი არის სპეციალური კაბელი, რომელიც გამოიყენება ინტერნეტის სწრაფ გადაცემისთვის. მისი შიგნით არის Core, ეს არის ცენტრალური ნაწილი, სადაც სინათლე მოძრაობს, რაც ინფორმაციას ატარებს. Cladding იცავს Core-ს, რომ სინათლე არ გამოვიდეს გარეთ, ხოლო Buffer Coating ამატებს დამცავ ფენას. Jacket კი ბოლო გარე საფარია, რომელიც კაბელს ზიანისგან იცავს. ეს კაბელი ძალიან სწრაფია და ხშირად გამოიყენება დიდ მანძილებზე, მაგალითად, ქალაქებს შორის.

**Copper Cable (სპილენძის კაბელი):** სპილენძის კაბელი გამოიყენება ინტერნეტისა და ტელეფონის კავშირისთვის, და მის შიგნით არის Copper Wire, ანუ სპილენძის ხაზები, რომლებიც ელექტროენერგიას ატარებენ. ამ ხაზებია Twisted Pairs-ებად, ანუ გაბრტყელებული წყვილებად, რაც ხმაურს ამცირებს. Pair Shields იცავს ყოველ წყვილს, ხოლო Overall Shield იცავს მთელ კაბელს გარე ზიანისგან. Jacket კი, როგორც ოპტიკურ კაბელში, გარე საფარია, რომელიც კაბელს დაცავს. ეს კაბელი ხშირად გამოიყენება სახლებში Wi-Fi-სთვის, მაგრამ ნაკლებად სწრაფია, ვიდრე ოპტიკური.

**სლაიდი 57**

ნახატზე ნაჩვენებია ორი ტიპის კაბელი — UTP და STP, რომლებიც გამოიყენება ქსელის კავშირებისთვის. ეს კაბელები ეფუძნება twisted pair ტექნოლოგიას, რაც ნიშნავს, რომ მათში ჩასმულია სხვადასხვა ფერის ხაზები, რომლებიც წყვილებად არის გახვეული. ამ გახვევამ ხელი შეუწყობს ხმაურის (ინტერფერენციის) შემცირებას, რაც მნიშვნელოვანია სტაბილური ინტერნეტის ან ქსელის მუშაობისთვის.

**### UTP კაბელი (Unshielded Twisted Pair)**

- \*\*მარტივად ახსნა:\*\* UTP არის უბრალო twisted pair კაბელი, რომელსაც არ აქვს დამატებითი დაცვა. ის შეიცავს მხოლოდ გახვეულ ხაზებს, რომლებიც გარკვეული ფერებით (მაგ., ლურჯი, მწვანე, ნარინჯისფერი) არის გამოყოფილი.

- \*\*მახასიათებლები:\*\* ეს კაბელი ხშირად გამოიყენება სახლში ან ოფისში Wi-Fi-ის ან LAN (ლოკალური ქსელის) დასაკავშირებლად. მისი უპირატესობაა ხელმისაწვდომი ფასი და მარტივი ინსტალაცია, მაგრამ ის ნაკლებად დაცულია გარე ხმაურისგან (მაგ., ელექტრომაგნიტური ინტერფერენცია).

**### STP კაბელი (Shielded Twisted Pair)**

- \*\*მარტივად ახსნა:\*\* STP არის twisted pair კაბელი, რომელსაც აქვს დამატებითი დაცვა . ამ ეკრანს (ჩვეულებრივ მეტალის ფენა) იცავს ხაზებს გარე ხმაურისგან.

- \*\*მახასიათებლები:\*\* ეს კაბელი უფრო ძლიერია და გამოიყენება იქ, სადაც საჭიროა მაღალი სიზუსტე ან დაცვა (მაგ., ინდუსტრიული გარემოები ან ადგილები, სადაც ბევრი ელექტრო მოწყობილობაა). ის უფრო ძვირია და მისი დაყენება რთულია, მაგრამ უზრუნველყოფს უკეთეს სიგნალს.

**სლაიდი 58**

ნახატზე ნაჩვენებია ორი განსხვავებული გზა, თუ როგორ უნდა დააკავშიროთ ქსელის კაბელები — ეს არის \*\*Straight Through\*\* და \*\*Crossover\*\*.ეს მეთოდები გამოიყენება იმისთვის, რომ ორი მოწყობილობა (მაგ., კომპიუტერი ან როუტერი) ერთმანეთთან დააკავშიროთ Ethernet კაბელის მეშვეობით. ნახატი გვიჩვენებს, თუ როგორ უნდა დაიწყონ ხაზები (pin-ები) ერთი მხრიდან და სად უნდა დასრულდნენ მეორე მხარეს. ამის მიხედვით, განვიხილოთ ორივე ვარიანტი მარტივად.

### Straight Through

- \*\*მარტივად ახსნა:\*\* ამ მეთოდით, ყველა ხაზი (pin) ერთნაირადაა დაკავშირებული ორივე მხარეს. მაგალითად, Pin 1 ერთ მხარეს Pin 1-თანაა დაკავშირებული მეორე მხარეს, Pin 2 — Pin 2-თან და ა.შ. ნახატზე ხაზები სწორად, ცვლილებების გარეშე, გადის.

- \*\*სად გამოიყენება:\*\* ეს მეთოდი გამოიყენება, როდესაც უნდა დააკავშიროთ განსხვავებული მოწყობილობები, მაგ., კომპიუტერი როუტერთან ან სვიჩთან. ამ შემთხვევაში, ერთი მოწყობილობა "გზავნის" მონაცემებს, ხოლო მეორე "მიიღებს" (მაგ., როუტერი გადაჰყავს ინტერნეტი კომპიუტერს).

**### Crossover**

- \*\*მარტივად ახსნა:\*\* ამ მეთოდით, ზოგიერთი ხაზი (pin) გადაკვეთილია. მაგალითად, Pin 1 ერთ მხარეს Pin 3-თანაა დაკავშირებული მეორე მხარეს, ხოლო Pin 2 — Pin 6-თან. ნახატზე ხაზები გადაკვეთილად გადის, რაც ცხადყოფს, რომ კავშირი განსხვავებულია.

- \*\*სად გამოიყენება:\*\* ეს მეთოდი გამოიყენება, როდესაც ორი ერთნაირი მოწყობილობა უნდა დააკავშიროთ ერთმანეთთან, მაგ., ორი კომპიუტერი პირდაპირ ერთმანეთთან. ამ შემთხვევაში, ორივე მოწყობილობა "გზავნის" და "მიიღებს" მონაცემებს, ამიტომ ხაზები უნდა იყოს გადაკვეთილი.

**სლაიდი 59**

**1. Single-Mode Fiber (SMF)**

* **ბირთვი**: ძალიან წვრილი (დაახლოებით 8–10 µm დიამეტრი)
* **როგორ გადის სინათლე**: მხოლოდ **ერთი გზა/მოდი** გადის ბირთვში
* **დაშორება**: დიდ მანძილზე მონაცემების გადაცემა შეუძლია (კილომეტრებით)
* **სიჩქარე**: მაღალი სიჩქარე, ნაკლები სიგნალის დაკარგვა
* **ლაზერი**: ძირითადად იყენებს **ლაზერულ** სინათლის წყაროს
* **გამოყენება**: დიდ დისტანციებზე, მაგ., ISP-ებში, ქალაქებს შორის კავშირებში

**2. Multi-Mode Fiber (MMF)**

* **ბირთვი**: უფრო სქელი (დაახლოებით 50–62.5 µm დიამეტრი)
* **როგორ გადის სინათლე**: **რამდენიმე გზა/მოდი** გადის ერთდროულად
* **დაშორება**: მოკლე დისტანციებზე უკეთესია (საშუალოდ 2 კმ-მდე ან ნაკლები)
* **სიჩქარე**: მაღალი სიჩქარე ახლო მანძილზე, მაგრამ შორს სიგნალი იშლება (modal dispersion)
* **ლამპა**: იყენებს **LED** სინათლის წყაროს
* **გამოყენება**: შენობებში, უნივერსიტეტებში, მცირე ქსელებში

**სლაიდი 60**

**1. Simplex (სიმპლექსი)**

* **მიმართულება**: მონაცემები გადადის **მხოლოდ ერთ მიმართულებით**
* **უკუ კავშირი არ არის**
* **მაგალითი**:
  + ტელევიზორი (სიგნალი მხოლოდ მაუწყებლიდან მოდის)
  + რადიო FM

**2. Half-Duplex (ნახევრად დუპლექსი)**

* **მიმართულება**: მონაცემები შეიძლება გადავიდეს **ორივე მიმართულებით**, მაგრამ **არა ერთდროულად**
* ჯერ ერთი მხარე აგზავნის, მერე მეორე
* **მაგალითი**:
  + ტოკი-ვოკი (გეტყვი → და მერე მომისმინე)
  + ორმხრივი რადიოები

**3. Full-Duplex (სრული დუპლექსი)**

* **მიმართულება**: მონაცემები გადადის **ორივე მიმართულებით ერთდროულად**
* **ყველაზე სწრაფი და ეფექტური რეჟიმი**
* **მაგალითი**:
  + ტელეფონი (ორივე ვლაპარაკობთ ერთდროულად)
  + თანამედროვე ინტერნეტი / Ethernet

**სლაიდი 61**

ნახატზე ნაჩვენებია ქსელის სხვადასხვა ტოპოლოგია, ანუ ის გზები, თუ როგორ არის დაკავშირებული კომპიუტერები ან სხვა მოწყობილობები ერთმანეთთან. ეს ტოპოლოგიები გამოიყენება ლოკალურ ქსელებში (LAN) ან სხვა კავშირგაბმულობის სისტემებში. ყოველი მათგანი განსხვავდება იმით, თუ როგორ არის განლაგებული მოწყობილობები და როგორ მუშაობს მონაცემების გადაცემა. განვიხილოთ ყველა მარტივად და დაწვრილებით.

**### 1. Point to Point**

- \*\*აღწერა:\*\* ორი მოწყობილობა (მაგ., ორი კომპიუტერი) პირდაპირ ერთმანეთთანაა დაკავშირებული ერთი ხაზით.

- \*\*მახასიათებლები:\*\* ეს ყველაზე მარტივი ტიპია, სადაც მონაცემები პირდაპირ გადადის ერთი მოწყობილობიდან მეორეზე. არ არის საჭირო დამატებითი მოწყობილობა (მაგ., როუტერი).

- \*\*სად გამოიყენება:\*\* მცირე და მარტივი კავშირებისთვის, მაგ., ორი კომპიუტერის შორის სახლში.

- \*\*მძიმე მხარე:\*\* თუ ხაზი დაირღვევა, კავშირი მთლიანად დაიშლება.

**### 2. Bus**

- \*\*აღწერა:\*\* ყველა მოწყობილობა დაკავშირებულია ერთი ცენტრალური ხაზის (ავტობუსის) გასწვრივ.

- \*\*მახასიათებლები:\*\* მონაცემები გადადის ხაზით, და ყველა მოწყობილობა იღებს მას, მაგრამ მხოლოდ განკუთვნილი მიიღებს. იაფია და მარტივია დასაყენებლად.

- \*\*სად გამოიყენება:\*\* მცირე ქსელებში, მაგ., ოფისში.

- \*\*მძიმე მხარე:\*\* თუ ცენტრალური ხაზი დაირღვევა, მთელი ქსელი გაითიშება.

**### 3. Ring (რგოლი)**

- \*\*აღწერა:\*\* მოწყობილობები დაკავშირებულია ერთმანეთთან წრიულად, ისე რომ ყველა მეზობელთანაა დაკავშირებული.

- \*\*მახასიათებლები:\*\* მონაცემები მოძრაობს ერთი მიმართულებით (ან ორმხრივად, თუ ორი რგოლია), ყოველი მოწყობილობა გადასცემს მას შემდეგს.

- \*\*სად გამოიყენება:\*\* ზოგიერთი ლოკალურ ქსელში, სადაც საიმედოობა მნიშვნელოვანია.

- \*\*მძიმე მხარე:\*\* თუ ერთი კავშირი დაირღვევა, შეიძლება მთელი რგოლი გაითიშოს.

### 4. Star (ვარსკვლავი)

- \*\*აღწერა:\*\* ყველა მოწყობილობა დაკავშირებულია ერთი ცენტრალური მოწყობილობასთან (მაგ., ჰაბი ან სვიჩი).

- \*\*მახასიათებლები:\*\* ცენტრალური მოწყობილობა აკონტროლებს მონაცემების გადაცემას. თუ ერთი მოწყობილობა გაითიშება, დანარჩენი ქსელი მუშაობს.

- \*\*სად გამოიყენება:\*\* ყველაზე გავრცელებულია ოფისებსა და სახლებში.

- \*\*მძიმე მხარე:\*\* თუ ცენტრალური მოწყობილობა დაირღვევა, მთელი ქსელი გაითიშება.

### 5. Tree (ხე)

- \*\*აღწერა:\*\* ეს არის Star-ის გაფართოება, სადაც ცენტრალური მოწყობილობები დაკავშირებულია ერთმანეთთან ხისებრად.

- \*\*მახასიათებლები:\*\* იძლევა მოქნილობას და შეიძლება გაიზარდოს ახალი მოწყობილობებით. ჰიერარქიულია.

- \*\*სად გამოიყენება:\*\* დიდ ქსელებში, მაგ., სკოლებში ან უნივერსიტეტებში.

- \*\*მძიმე მხარე:\*\* რთულია მართვა, თუ ცენტრალური კვანძი დაირღვევა.

### 6. Mesh (ქსელი)

- \*\*აღწერა:\*\* ყველა მოწყობილობა დაკავშირებულია ერთმანეთთან, რაც ქსელისებრ გამოსახულებას ქმნის.

- \*\*მახასიათებლები:\*\* ძალიან საიმედოა, რადგან თუ ერთი კავშირი დაირღვევა, მონაცემები სხვა გზით გადადის. შეიძლება იყოს სრული (ყველა-ყველასთან) ან ნაწილობრივი.

- \*\*სად გამოიყენება:\*\* კრიტიკულ სისტემებში, მაგ., სამხედრო ან ფინანსურ ქსელებში.

- \*\*მძიმე მხარე:\*\* ძალიან ძვირია და რთულია დასაყენებლად.

### 7. Hybrid (ჰიბრიდული)

- \*\*აღწერა:\*\* ეს არის სხვადასხვა ტოპოლოგიის (მაგ., Star და Ring) შეერთება ერთ სისტემად.

- \*\*მახასიათებლები:\*\* მოქნილია და შეიძლება გამოყენებულ იქნას სხვადასხვა მოთხოვნებისთვის. აერთიანებს სხვა ტიპების უპირატესობებს.

- \*\*სად გამოიყენება:\*\* დიდ და რთულ ქსელებში, მაგ., კორპორაციებში.

- \*\*მძიმე მხარე:\*\* რთულია დაგეგმვა და მოვლა.