

Протокол UDP

Дисциплина: Сетевые технологии

Мишина Анастасия Алексеевна

Содержание

1 Введение	5
2 Протокол UDP	6
2.1 Состав пакета UDP	6
2.2 Псевдозаголовки	8
3 Практическое применение UDP	10
4 Преимущества и недостатки UDP	17
5 Заключение	19
Список литературы	20

Список иллюстраций

2.1	Структура пакета UDP	6
2.2	Структура псевдозаголовка пакета UDP	8
3.1	Сервер	11
3.2	Клиент	11
3.3	Wireshark IPv4	12
3.4	Физический уровень	13
3.5	Канальный уровень	13
3.6	Сетевой уровень	14
3.7	Транспортный уровень	14
3.8	Сервер	15
3.9	Клиент	15
3.10	Wireshark IPv6	16

Список таблиц

1 Введение

Протокол UDP (User Datagram Protocol) — это один из протоколов транспортного уровня, применяемых в сетях передачи данных. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в RFC 768 (документ, определяющий протокол User Datagram Protocol (UDP)). В отличие от TCP (Transmission Control Protocol), протокол UDP является легковесным и обеспечивает минимальную надстройку над сетевым уровнем. Он не гарантирует, что отправленные данные достигнут получателя, и не предусматривает механизмов для восстановления утраченных пакетов. UDP идеально подходит для приложений, где скорость передачи данных важнее, чем их полная сохранность, например, для видеоконференций, онлайн-игр и потокового воспроизведения мультимедиа.

UDP функционирует на транспортном уровне модели OSI и предоставляет лишь базовые возможности для обмена данными между приложениями. Он не требует предварительного установления соединения между отправителем и получателем (connectionless communication), что ускоряет передачу, но снижает надежность. Один узел сети просто отправляет пакеты, адресуя их другому узлу. Отправитель не знает ничего о том, готов ли получатель к приёму пакетов, и вообще, существует ли этот получатель. Отправитель также не ждёт какого-либо подтверждения о том, что получатель принял предыдущие пакеты.

2 Протокол UDP

2.1 Состав пакета UDP

Пакеты, передаваемые с помощью UDP, иногда называются датаграммами. Этот термин обычно используется тогда, когда важно подчеркнуть, что пакет передаётся без установки соединения. Рассмотрим состав пакета UDP (рис. 2.1).

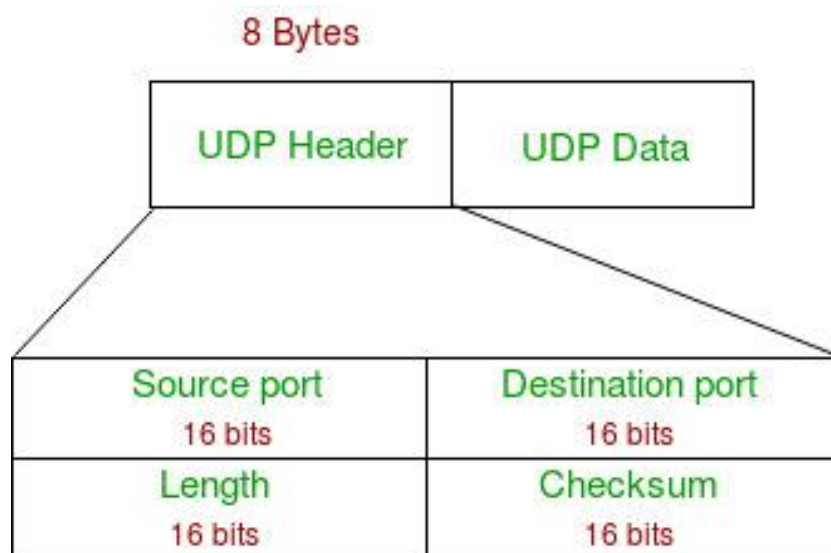


Рис. 2.1: Структура пакета UDP

UDP датаграммы состоят из заголовка и раздела с данными [1]. Заголовок пакета UDP состоит из 8 байт, в то время как для TCP он может варьироваться от 20 до 60 байт. UDP-порты используются для идентификации приложений или служб, которые отправляют или получают данные. Номера портов представлены 16-битными беззнаковыми целыми числами в диапазоне от 0 до 65535.

Приведу некоторые распространённые номера UDP-портов и связанные с ними приложения или службы:

- 53 — DNS (система доменных имён);
- 67/68 — DHCP (протокол динамической конфигурации хоста) сервер/клиент;
- 69 — TFTP (простой протокол передачи файлов);
- 123 — NTP (сетевой протокол времени);
- 161/162 — SNMP (простой протокол управления сетью).

Вернемся к рассмотрению состава заголовка. Он включает в себя:

- Исходный порт: поле длиной 2 байта. В этом поле указывается номер порта отправителя. Предполагается, что это значение задаёт порт, на который при необходимости будет посылаться ответ. В противном же случае значение будет равным 0.
- Порт назначения: поле длиной 2 байта. Это поле обязательно и содержит порт получателя.
- Длина датаграммы: длина пакета UDP, включает в себя длину заголовка и данных. Минимальная длина равна длине заголовка — 8 байт. Теоретически, максимальный размер поля — 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 — 65507 (помимо 8 байт на UDP-заголовок требуется ещё 20 на IP-заголовок). На практике следует учитывать, что если длина IPv4 пакета с UDP будет превышать 1500 байт, то отправка может вызвать фрагментацию пакета и он может быть не доставлен, если промежуточные маршрутизаторы или конечный хост не будут поддерживать фрагментированные IP пакеты. Для того чтобы быть уверенным, что пакет будет принят любым хостом, размер данных в UDP не должен превышать: (минимальная длина IP пакета) — (Max IP Header Size) — (UDP Header Size) = 576 — 60 — 8 = 508 байт. IPv6 пакеты UDP могут иметь больший размер. Максимальное значение составляет 4 294 967 295 байт ($2^{32} - 1$), из которых 8 байт

соответствуют заголовку, а остальные 4 294 967 287 байт — данным.

- Контрольная сумма: поле длиной 2 байта. Контрольная сумма используется для проверки заголовка и данных на ошибки. Если сумма не сгенерирована передатчиком, то поле заполняется нулями.

Порт отправителя и контрольная сумма необязательны к использованию в IPv4, в то время как в IPv6 необязателен только порт отправителя.

2.2 Псевдозаголовки

UDP-заголовок не содержит информации об адресе отправителя и получателя, поэтому даже при совпадении порта получателя нельзя с точностью сказать, что сообщение пришло в нужное место. Для проверки того, что UDP-сообщение достигло пункта своего назначения, вычисляется контрольная сумма с использованием дополнительного псевдозаголовка (рис. 2.2). Псевдозаголовок не включается в UDP-сообщение.



Рис. 2.2: Структура псевдозаголовка пакета UDP

Вначале идут поля IP-адрес источника (длина 32 бит) и IP-адрес получателя (длина 32 бит). Далее идёт зарезервированное поле (длина 8 бит), заполненное нулями. Поле Протокол (длина 8 бит) идентифицирует протокол из заголовка пакета IP. Для UDP это значение равно 17 (00010001 в двоичном виде, 0x11 —

в шестнадцатеричном). Далее идёт поле Длина UDP (длина 16 бит). Защита заголовка IP несколько избыточна и делает протокол UDP (впрочем, как и TCP) неотделимым от протокола IP, хотя это и позволяет провести двойную проверку датаграмм IP, поступивших для заданного получателя [2].

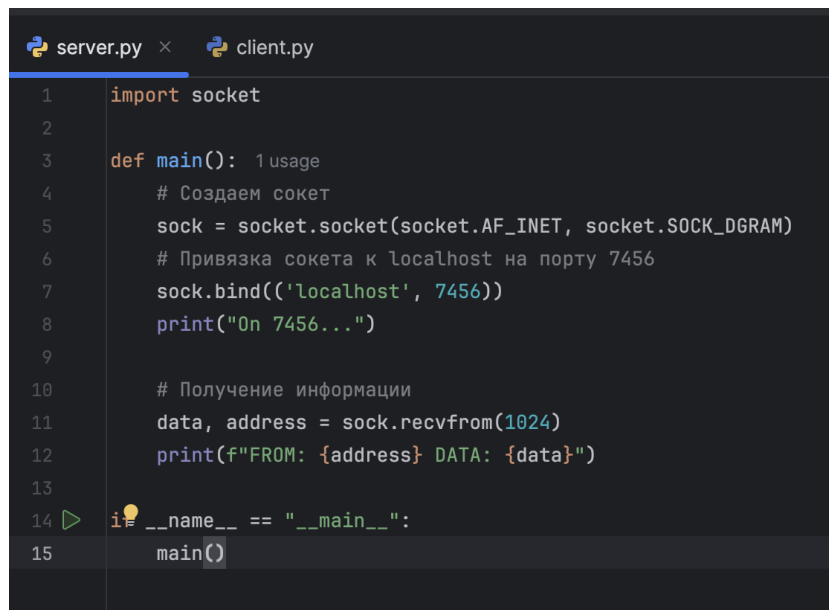
Контрольная сумма считается перед отправлением сообщения и при его получении (получатель составляет свой псевдозаголовок, используя адрес хоста, с которого пришло сообщение, и собственный адрес, а затем считает контрольную сумму).

Если контрольная сумма совпадает, значит, пакет достиг целевого узла назначения, а также правильного порта протокола на этом узле. Если контрольная сумма в полученном пакете равняется нулю, то считается, что передающий уровень UDP её не вычисляет, и данные не защищены.

3 Практическое применение UDP

Для рассмотрения протокола UDP воспользуемся языком программирования python. Нам понадобится написать минимальный функционал для сервера и клиента. Воспользуемся модулем socket, который предоставляет доступ к низкоуровневым сетевым интерфейсам. Он позволяет создавать и управлять сокетами (название программного интерфейса для обеспечения обмена данными между процессами).

Код для сервера (рис. 3.1). Создаем сокет: AF_INET - используем IPv4, SOCK_DGRAM - сокет будет использовать протокол UDP. Сокет привязывается к адресу localhost (что означает, что он будет доступен только на локальном компьютере) и порту 7456. Это позволяет сокету прослушивать входящие сообщения, отправленные на этот порт. Метод recvfrom(1024) блокирует выполнение программы до тех пор, пока не поступит сообщение. Максимальный размер получаемых данных составляет 1024 байта. Когда данные приходят, они сохраняются в переменной data, а адрес отправителя — в переменной address.



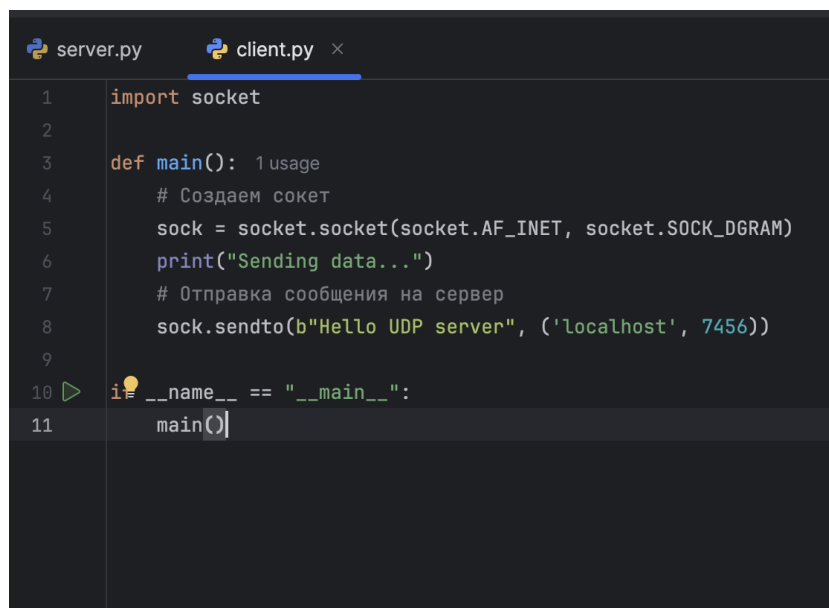
```

server.py x client.py
1  import socket
2
3  def main(): 1 usage
4      # Создаем сокет
5      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6      # Привязка сокета к localhost на порту 7456
7      sock.bind(('localhost', 7456))
8      print("On 7456...")
9
10     # Получение информации
11     data, address = sock.recvfrom(1024)
12     print(f"FROM: {address} DATA: {data}")
13
14     if __name__ == "__main__":
15         main()

```

Рис. 3.1: Сервер

Код для клиента (рис. 3.2). Аналогично создаем сокет и отправляем сообщение с текстом “Hello UDP server”.



```

server.py client.py x
1  import socket
2
3  def main(): 1 usage
4      # Создаем сокет
5      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6      print("Sending data...")
7      # Отправка сообщения на сервер
8      sock.sendto(b"Hello UDP server", ('localhost', 7456))
9
10     if __name__ == "__main__":
11         main()

```

Рис. 3.2: Клиент

Запускаем wireshark и устанавливаем прослушивание на Loopback: lo0. Loopback - это виртуальный сетевой интерфейс, который используется для

связи внутри самого устройства. В большинстве операционных систем, включая Unix-подобные системы (такие как Linux и macOS), а также Windows, loopback-интерфейс имеет IP-адрес 127.0.0.1.

Запускаем оба скрипта и в терминале видим следующее сообщение: FROM: ('127.0.0.1', 53439) DATA: b'Hello UDP server'. Это означает, что сообщение с нашим текстом было отправлено с ip-адреса 127.0.0.1 (localhost) и с порта 53439 - каждый раз, когда приложение или процесс открывает сокет для отправки данных, ему назначается случайный порт (в данном случае 53439) для этой сессии.

Заходим в Wireshark, фильтруем пакеты по порту - `udp.port == 7456` и видим строку с протоколом UDP. Справа можем сразу наблюдать наше сообщение (рис. 3.3).

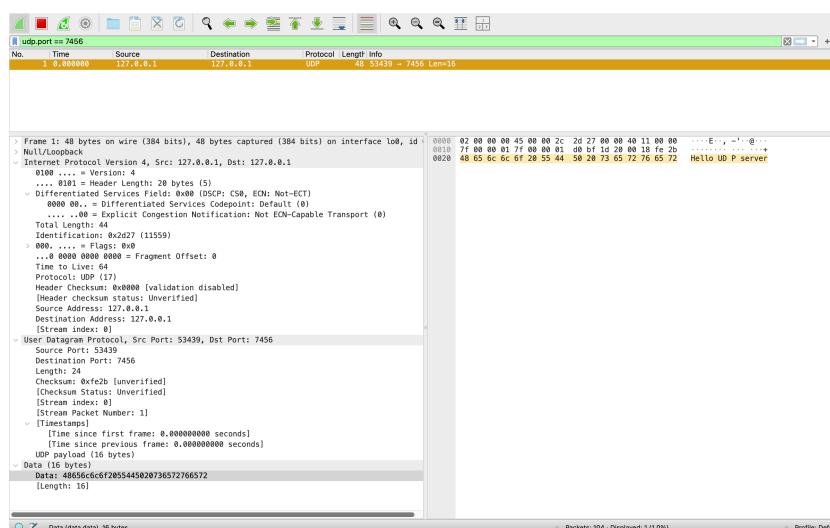


Рис. 3.3: Wireshark IPv4

Рассмотрим подробнее информацию по пакету [3]:

- Фрейм: Обзор фрейма данных физического уровня (рис. 3.4). Например, тип упаковки - Loopback, номер кадра - 1, длина кадра - 48 байт, длина байта захвата - 48 байт.

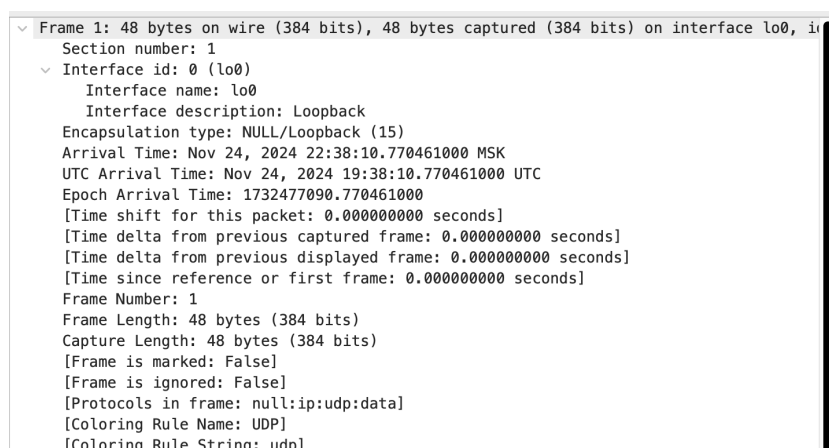


Рис. 3.4: Физический уровень

- Null/Loopback канального уровня (рис. 3.5). Это поле содержит значение «семейства адресов»/«семейства протоколов» для протокола, работающего на канальном уровне, например AF_INET для IPv4 и AF_INET6 для IPv6. AF_INET равен 2 во всех операционных системах на базе BSD, поскольку он был введен одновременно с выпуском версий BSD с сетевыми возможностями; однако AF_INET6, к сожалению, имеет разные значения в {NetBSD, OpenBSD, BSD/OS}, {FreeBSD, DragonFlyBSD} и {Darwin/macOS}, поэтому пакет IPv6 может иметь заголовок канального уровня со значением 24, 28 или 30.



Рис. 3.5: Канальный уровень

- Интернет-протокол версии 4: информация заголовка IP-пакета сетевого уровня (рис. 3.6). Общая длина ip-пакета - 44 байта, исходный ip-адрес - 127.0.0.1, ip-адрес получателя 127.0.0.1.

```

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 44
  Identification: 0x2d27 (11559)
  000. .... = Flags: 0x0
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 127.0.0.1
  Destination Address: 127.0.0.1
  [Stream index: 0]

```

Рис. 3.6: Сетевой уровень

- В разделе User Datagram Protocol можно заметить заголовок и те поля, что обсуждались ранее. Это транспортный уровень (рис. 3.7). Порт источника - 53439, порт получателя (сервер) - 7456, длина - 24 байта (длина заголовка 8 + длина оставшихся данных в пакете), контрольная сумма - 0xfe2b.

```

> Frame 1: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface lo0, id 1
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 53439, Dst Port: 7456
  Source Port: 53439
  Destination Port: 7456
  Length: 24
  Checksum: 0xfe2b [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Stream Packet Number: 1]
  [Timestamps]
    [Time since first frame: 0.000000000 seconds]
    [Time since previous frame: 0.000000000 seconds]
  UDP payload (16 bytes)
  Data (16 bytes)
    Data: 48656c6c662055445020736572766572
    [Length: 16]

```

Рис. 3.7: Транспортный уровень

Теперь попробуем повторить эксперимент, но уже не с IPv4, а с IPv6. Для этого в сервере (рис. 3.8) и клиенте (рис. 3.9) меняем AF_INET на AF_INET6.

```
server.py x client.py
1 import socket
2
3 def main(): 1 usage
4     # Создаем сокет
5     # sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6     sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
7     # Привязка сокета к localhost на порту 7456
8     sock.bind(('localhost', 7456))
9     print("On 7456...")
10
11     # Получение информации
12     data, address = sock.recvfrom(1024)
13     print(f"FROM: {address} DATA: {data}")
14
15 if __name__ == "__main__":
16     main()
```

Рис. 3.8: Сервер

```
server.py x client.py
1 import socket
2
3 def main(): 1 usage
4     # Создаем сокет
5     # sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6     sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
7     print("Sending data...")
8     # Отправка сообщения на сервер
9     sock.sendto(b"Hello UDP server", ('localhost', 7456))
10
11 if __name__ == "__main__":
12     main()
```

Рис. 3.9: Клиент

Запускаем захват в Wireshark, затем запускаем сервер и клиент. Видим следующее сообщение в терминале: FROM: (::1, 49389, 0, 0) DATA: b'Hello UDP server'. ::1 — это IPv6-адрес, который является локальным адресом (аналогично 127.0.0.1 в IPv4), он используется для обозначения "localhost". 49389 — это номер порта, с

которого был отправлен пакет. 0, 0: первое значение это flow info (информация о потоке). Это поле связано с метками потока (flow labels) в IPv6, которые могут быть полезны для маршрутизации пакетов, принадлежащих одному потоку данных. Flow label — это 20-битное значение, встроенное в заголовок IPv6, которое может использоваться для приоритизации трафика или других целей. В большинстве случаев, если потоковая информация не используется, это значение равно 0. Второе значение 0 это scoreid, то есть индекс зоны (zone index), который указывает конкретный сетевой интерфейс, связанный с адресом отправителя.

Заходим в Wireshark, фильтруем пакеты по порту - `udp.port == 7456` и видим строку с протоколом UDP (рис. 3.10).

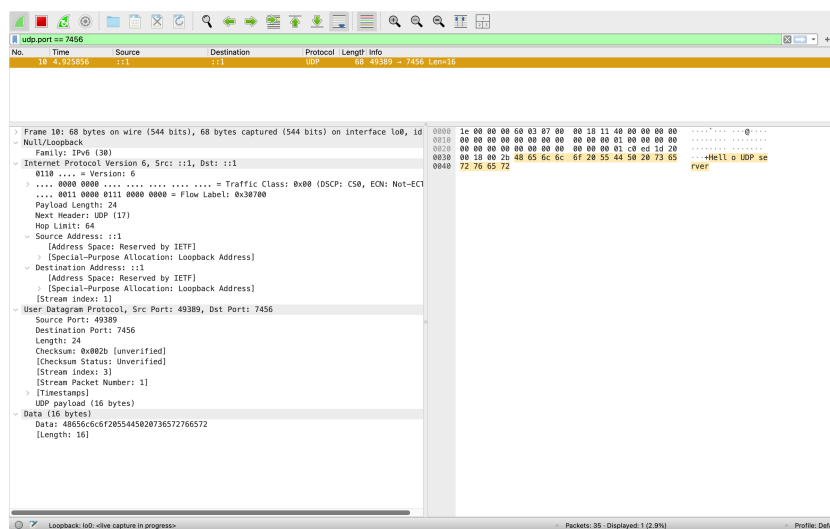


Рис. 3.10: Wireshark IPv6

На канальном уровне видим значение “семейства протоколов” равному 30, что указывает на использование IPv6. Посмотрим на различия на транспортном уровне протокола UDP. Видим, что блок с данными весит по 16 байт в обоих случаях, порт назначения и длина пакета вместе с данными также совпадают. Различия есть в контрольных суммах, так как у IPv4 и IPv6 заголовки имеют разный формат. Поскольку в случае IPv6 псевдозаголовков значительно больше и содержит более длинные адреса, итоговая контрольная сумма будет отличаться, даже если полезная нагрузка (данные UDP) идентична.

4 Преимущества и недостатки UDP

Преимущества UDP:

- Скорость: UDP работает быстрее, чем TCP, потому что ему не нужно устанавливать соединение и обеспечивать надёжную передачу данных.
- Меньшая задержка: поскольку не требуется установка соединения, задержка меньше, а время отклика быстрее.
- Простота реализации: Он не требует сложных механизмов для управления потоком данных и контроля ошибок, что упрощает разработку и уменьшает нагрузку на систему. Это особенно важно для разработчиков, которые хотят быстро и эффективно внедрить сетевые функции в свои приложения. Простота UDP позволяет сосредоточиться на основной логике приложения, а не на сложных механизмах транспортного уровня.
- Поддержка широковещательной и многоадресной передачи: UDP поддерживает широковещательную (broadcast) и многоадресную (multicast) передачу данных, что позволяет отправлять данные сразу нескольким получателям. Это полезно для приложений, таких как видеоконференции и онлайн-трансляции. Широковещательная передача позволяет отправлять данные всем устройствам в сети, а многоадресная передача — только определенной группе устройств, что делает UDP гибким инструментом для различных сценариев использования.
- Меньшая нагрузка на сеть: Так как UDP не использует механизмы подтверждения и повторной передачи данных, он создает меньшую нагрузку на сеть. Это особенно важно в условиях ограниченной пропускной способно-

сти, где каждый бит на счету. UDP позволяет передавать данные с минимальными накладными расходами, что делает его подходящим для приложений с высокой интенсивностью передачи данных, таких как стриминг видео и аудио.

Недостатки UDP:

- Отсутствие надёжности: UDP не гарантирует доставку пакетов или порядок их доставки, что может привести к потере или дублированию данных.
- Отсутствие контроля ошибок: UDP не включает механизмов для обнаружения и исправления ошибок. Если данные повреждены во время передачи, они будут доставлены получателю в поврежденном виде.
- Отсутствие управления потоком: UDP не предоставляет механизмов для управления потоком данных. Это может привести к перегрузке сети или получателя, особенно в условиях высокой нагрузки. В отличие от TCP, который использует механизмы управления потоком для предотвращения перегрузки, UDP отправляет данные без учета состояния сети или получателя.
- Использование UDP делает систему уязвимой к атакам: UDP уязвим для атак типа DoS (отказ в обслуживании), когда злоумышленник может заполнить сеть UDP-пакетами, перегрузив её и вызвав сбой.
- Ограниченная поддержка в некоторых сетях: Некоторые сети и маршрутизаторы могут ограничивать или блокировать UDP-трафик из-за его потенциальной ненадежности и отсутствия механизмов контроля. Это может ограничить использование UDP в определенных сетевых условиях.

5 Заключение

Протокол пользовательских дейтаграмм (UDP) - важный протокол транспортного уровня в наборе интернет-протоколов, известный своей скоростью и эффективностью из-за отсутствия подключения к интернету и легкого дизайна. Хотя UDP не обладает стабильностью TCP и функциями проверки ошибок, он используется в приложениях, которым требуется низкая задержка и производительность в реальном времени, таких как потоковая передача, онлайн-игры и поиск в DNS. Его простота и поддержка широковещательных и многоадресных трансляций делают его полезным инструментом для специализированных приложений, несмотря на его уязвимость к потере данных и перегрузке сети. Пользуясь UDP, приложение само несёт ответственность за коррекцию ошибок.

Список литературы

1. User Datagram Protocol (UDP). URL: <https://www.geeksforgeeks.org/user-datagram-protocol-udp/>; [Электронный ресурс], 2024-09-27.
2. Кулябов Д.С., Королькова А.В. Архитектура и принципы построения современных сетей и систем телекоммуникаций: Учеб. пособие. Москва: РУДН, 2008. 309 с.
3. Сандерс К. АНАЛИЗ ПАКЕТОВ Практическое руководство по использованию Wireshark и tcpdump для решения реальных проблем в локальных сетях. Санкт-Петербург: Диалектика, 2019. 448 с.