NACKADEMIN

Android-app som visar aktuell tid, synkroniserad med en NTP-server

Jimmy kroneld

Jimmy.kroneld@yh.nackademin.se

Innehållsförteckning

Vad är en NTP-Server:

Network Time Protocol (NTP) är en standard Internet Protocol (IP) för att synkronisera klockor på datorer i ett nätverk."

NTP kommunicerar med User Datagram Protocol (UDP), port 123.

NTP kan fungera på flera sätt. Den vanligaste konfigurationen är att fungera i unicast-läge, eller klient-server. Här sänder en klient en begäran i form av ett paket till en server, som svarar med ett tidsstämpelpaket. Varje paket har ursprungs-, mottagnings- och sändningstidsstämplar så att nätverksfördröjningar kan beräknas. Detta gör att klienter kan synkronisera så nära som möjligt med serverns klocka.

NTP använder ett hierarkiskt system för att skapa bättre kommunikation, vilket kallas för "clock strata". Dessa strata innefattar:

Stratum 0 är enheter så som atomur, GPS-klockor och andra former av radiobaserade klockor. Detta är högsta nivån i hierarkin.

Stratum 1 kallas även för primära NTP-servrar. Stratum 1-servrar hämtar sin tid från stratum 0-enheter via direktkopplingar, så som RS-232.

Stratum 2 kallas även för sekundära NTP-servrar. Dessa hämtar sin tid från stratum 1-servrarna, det vill säga från de primära servrarna.

Stratum 3 hämtar sin tid från stratum 2-servrar och fungerar på samma sätt som en stratum 2-server i övrigt. Andra servrar kan i sin tur hämta tid från en stratum 3-server.

För varje nivå minskar exaktheten något, dock knappt märkbart för normal användning.

UdpSntpClient

Motivering för egen UDP-klass för tidssynkronisering:

För tidssynkronisering i min app valde jag att undvika tredjepartsbibliotek. I stället skapade jag en egen lösning baserad på UDP (User Datagram Protocol) av följande anledningar:

- **Oberoende**: Jag ville inte vara beroende av externa bibliotek, vilket kan medföra kompatibilitetsproblem. Exempel på detta är när olika versioner av samma bibliotek krockar eller när ett bibliotek inte stöder alla plattformar.
- **Kontroll**: Jag ville ha full kontroll över funktionaliteten och kunna anpassa den efter mina behov.
- **Utmaning**: Jag ville utmana mig själv och införskaffa mig en djupare förståelse för NTP server och dess implementation.

För att uppnå detta i Java använde jag:

- **DatagramPacket**: Ett paket som representerar data som kan sändas eller mottas, innehållande både själva datan och nätverksinformationen.
- **DatagramSocket**: En slutpunkt för att sända eller ta emot **DatagramPacket**s.

Genom dessa klasser kunde jag skapa en robust tidssynkroniseringslösning utan att behöva förlita mig på externa bibliotek.

DatagramPacket:

DatagramPacket: Data som skickas är inkapslad i en enhet som kallas för Datagram, man kan skapa ett DatagramPacket objekt genom att använda sig utav olika ”constructors” nedan är två olika:

Constructor 1:

DatagramPacket(byte[] buf, int length);

Constructor 2:
DatagramPacket(byte[] buf, int length, InetAddress address, int port);

Förstnämnda konstruktorn (Constructor 1) används främst för när man ska ta emot data, exempel på ett scenario när man hade använt denna är :

Lyssna på en specifik port: När man har en klient som är inställd på att lyssna på en specifik port, innebär det att den är redo att ta emot data på den porten. I detta scenario skapar man ett DatagramPacket med denna konstruktor för att ta emot dessa meddelanden. Eftersom man inte vet på förhand varifrån datan kommer (vilken IP-adress eller port), behöver man bara ange en byte-array och dess längd som ska fyllas med den inkommande datan.

Andranämnda Konstruktorn (Constructor 2) används när man vill skicka data till en specifik adress och port. Här specificerar man både data (i form av en byte-array), längden av datan, destinationens IP-adress och portnummer. Exempel på ett scenario när man hade använt denna:

Skicka en förfrågan: Till exempel när man skickar en tidsförfrågan till en NTP-server.

Genom att förstå dessa två konstruktorer och när man ska använda dem kan man effektivt hantera UDP-kommunikation i Java

DatagramSocket:

DatagramSocket är en klass i Java som används för att skicka och ta emot datagram-paket över ett nätverk med hjälp av UDP-protokollet. För att kunna skicka ett datagram måste en DatagramSocket skapas.
Den erbjuder metoder för att binda till en lokal port, skicka och ta emot datagram-paket, samt stänga anslutningen.
DatagramSocket metoder:

- *socket.close()*
- *socket.receive(packet)*
- *socket.send(packet);*

Skapar man utan några argument så kommer så binds socketen till en ledig port som systemet väljer

- *DatagramSocket socket = new DatagramSocket();*

Om man vill välja port själv så anger man det som argument.

- *DatagramSocket socket = new DatagramSocket(123);*

NTP Datapaket struktur:

Detta är NTP-protokollets datastruktur. Även om man använder SNTP, som är en förenklad version av NTP, är det fortfarande viktigt att förstå denna struktur.

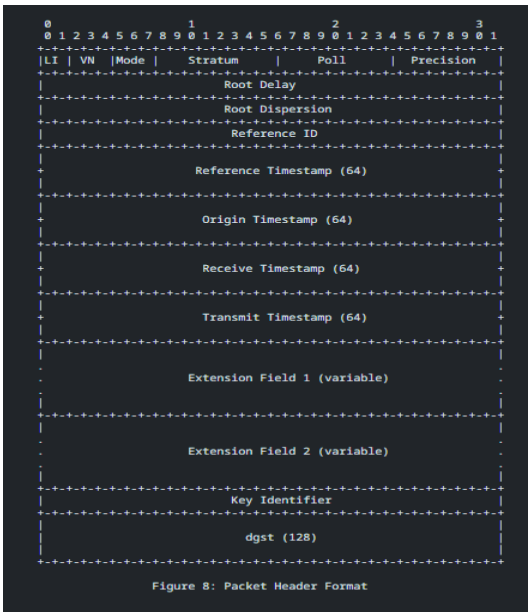


Figure 8: Packet Header Format

Många element i denna struktur kan specificeras av en utvecklare, men i SNTP kan vissa delar inte användas eller hanteras på ett annorlunda sätt.
NTP och SNTP använder samma datapaketsstruktur, som den du ser i skärmdumpen. Men när det gäller synkroniseringslogiken skiljer de sig åt.
NTP har mer avancerade funktioner för att garantera exakt tidhållning över längre perioder och under varierande nätverksförhållanden. Till exempel kan du som kodare specificera vissa fält i NTP-paketet, men i SNTP kanske vissa delar inte används.

Definiera NTP request packet:

	Bits 0-7	Bits 8-15	Bits 16-23	Bits 24-31
Bytes 0-3	LI | VN | Mode	Stratum (8 bits)	Poll (8 bits)	Precision (8 bits)
Bytes 4-7	Root delay (32 bits)			
Bytes 8-11	Root dispersion (32 bits)			
Bytes 12-15	Reference identifier (32 bits)			
Bytes 16-19	Reference timestamp (64 bits)			
Bytes 20-23				
Bytes 24-27	Originate timestamp (64 bits)			
Bytes 28-31				
Bytes 32-35	Receive timestamp (64 bits)			
Bytes 36-39				
Bytes 40-43	Transmit timestamp (64 bits) in format :- whole seconds (32 bits) : fraction of second (32 bits)			
Bytes 44-47				
Bytes 48-51	Authenticator (optional, 128 bits)			
Bytes 52-55				
Bytes 56-59				
Bytes 60-63				

Som i bilden kan man se att den första raden (bytes 0–3) är uppdelad i 4 segment. Dessa segment representerar olika delar av NTP-protokollets specifikation för det första bytet.

```
//Define NTP request packet - 48 bytes (all zeros)
byte[] ntpRequest = new byte[48];
ntpRequest[0] = 0x23; // LI, Version, Mode
```

Man skapar en NTP request packet som innehåller 48 bytes(är satt i första hand till 0)
Sedan sätter man första byten till ’0x23’ och är en specificerar NTP protokollet.

Värdet ’0x23’ för det första bytet i NTP-request paketet kommer från NTP-protokollets specifikation. I NTP-meddelandeformatet:

- Bitar 0–1 (de två yttersta bitarna till vänster) representerar **Leap Indicator (LI)**. 00 betyder "ingen skottsekund".
- Bitar 2–4 (de nästa tre bitarna) representerar **Version Number (VN)**. I detta sammanhang indikerar 100 version 4 av NTP-protokollet.
- Bitar 5–7 (de tre yttersta bitarna till höger) representerar **Mode**. Värdet 011 i detta sammanhang indikerar klientläge.

När du kombinerar dessa bitvärden får du 00100011 i binärt, vilket är ’0x23’ i hexadecimalt.

Alltså, genom att ställa in det första bytet av paketet till ’0x23’ indikeras:

- *Ingen justering av skottsekunden ska göras.*
- *Du använder version 4 av NTP-protokollet.*
- *Paketet är en klientbegäran till en NTP-server.*

Detta är ett standardiserat sätt att konstruera NTP-paketet för att skicka en begäran från en klient till en server. De exakta detaljerna, inklusive användningen av ’0x23’ och de olika lägena, specificeras

	Bits 0-7	Bits 8-15	Bits 16-23	Bits 24-31
Bytes 0-3	LI | VN | Mode	Stratum (8 bits)	Poll (8 bits)	Precision (8 bits)
Bytes 4-7	Root delay (32 bits)			
Bytes 8-11	Root dispersion (32 bits)			
Bytes 12-15	Reference identifier (32 bits)			
Bytes 16-19	Reference timestamp (64 bits)			
Bytes 20-23				
Bytes 24-27	Originate timestamp (64 bits)			
Bytes 28-31				
Bytes 32-35	Receive timestamp (64 bits)			
Bytes 36-39				
Bytes 40-43	Transmit timestamp (64 bits) in format :- whole seconds (32 bits) : fraction of second (32 bits)			
Bytes 44-47				
Bytes 48-51	Authenticator (optional, 128 bits)			
Bytes 52-55				
Bytes 56-59				
Bytes 60-63				

i NTP-protokollspecifikationen, specifikt i *RFC 5905.*

Timestamp:

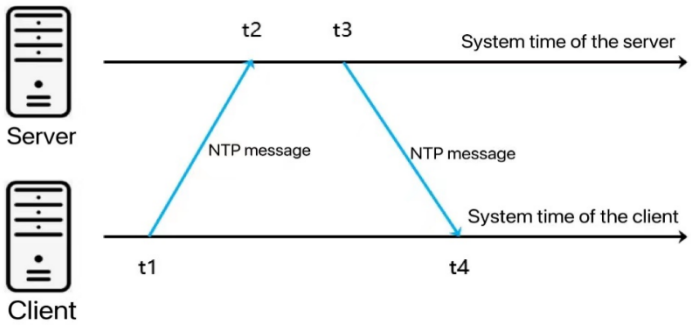
NTP-tidsstämpeln börjar räkna från 1900 och Unix-tidsstämpeln börjar räkna från 1970, måste du justera för denna skillnad genom att subtrahera 2208988800L. För att konvertera tidsstämpeln till millisekunder multipliceras den sedan med 1000L. När detta är gjort har du en tidsstämpel i Unix-format. När du har UNIX-tiden kan du omvandla den till ett Date-objekt eller något annat tidsformat du föredrar.

När du vill hämta tidsstämpeln från en NTP-server använder du en buffer för att lagra svaret. Denna buffer innehåller flera bitar av information, inklusive tidsstämpel.

Det finns fyra olika tidstämplar i paketet(som man kan se på bilden ovan):

1. Reference timestamp(64 bits)
2. Originate timestamp(64 bits)
3. Receive timestamp(64 bits)
4. Transmit timestamp(64 bits)

Dessa var för sig delas upp i två delar, 4 bytes eller 32bits(sekunder) och 4 bytes eller 32bits(bråkdelar av en sekund)



- **t1** = 24-27 -- 28-31 (*Original timestamp*) ***time at the time when the request is sent according to the client clock***
- **t2** = 32-35 -- 36-39 (*Receive timestamp*) **time at the time when the server received the request according to its clock**
- **t3** = 40-43 -- 44-47 (*Transmit timestamp*) **time to the moment when the answer is returned according to the server clock**
- **t4** = 16-19 -- 20-23 (*Reference timestamp*) **time to the moment when the answer is received by the client according to his clock**

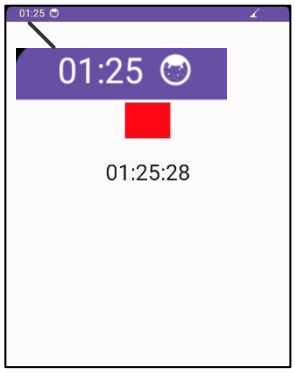
Offset (Theta)

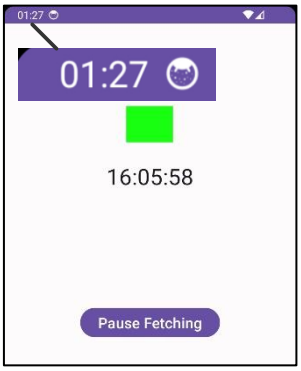
Offset beskriver tidsdifferensen mellan klientens och serverns klockor. Genom att beräkna offset kan man justera och synkronisera tiderna så att båda enheterna matchar varandra.

- *Offset = (t2 - t1) + (t3 - t4)/ 2*

Systemtid + Offset

I applikationen så visas systemtiden med adderad tidsskillnad(Offset), för att illustrera detta har jag





inkluderat två bilder. Bild 1 visar systemtiden när internet är nere och ingen synkronisering kan ske. Bild 2 visar systemtiden plus offset när internet är uppkopplat och synkronisering har skett.

På Bild 1 ser vi systemtiden som visas utan någon justering. Detta är den tid som klientens klocka visar utan att ha synkroniserat med en NTP-server. Om internet är nere eller om servern inte kan nås, kommer systemtiden att förbli oförändrad.

På Bild 2 ser vi systemtiden plus offset. När klienten har kontakt med NTP-servern och har mottagit den aktuella tiden, beräknas offset enligt formeln ovan. Denna offset läggs sedan till systemtiden för att justera klockan så att den matchar serverns tid. På så sätt synkroniseras klientens klocka med servern, och systemtiden blir korrekt.

Referenser:

Editorial Staff, 'NTP - An Introduction To Network Time Protocol', TimeTools Ltd, n.d., *https://timetoolsltd.com/ntp/what-is-ntp/*.

Structure of the NTP Data packet', Meinberg Funkuhren GmbH & Co KG, n.d., *https://www.meinbergglobal.com/english/info/ntp-packet.htm*.

Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch. 2010. "Network Time Protocol Version 4: Protocol and Algorithms Specification". Internet Engineering Task Force. *https://datatracker.ietf.org/doc/html/rfc5905*

Nam Ha Minh, 'Java UDP Client Server Program Example', CodeJava, 18 July 2019, *https://www.codejava.net/java-se/networking/java-udp-client-server-program-example*.

Netnod, 'Swedish Distributed Time Service', Netnod, n.d., *https://www.netnod.se/swedish-distributed-time-service*.