Universidade Federal de Pernambuco

Centro de Informática

Mestrado em Ciência da Computação

"Clock Synchronization in Computer Networks with Quality of Service"

By:

Arthur de Castro Callado

This dissertation has been submitted to the computer science center of the Federal University of Pernambuco as a partial requirement to obtain the degree of Master in Computer Science.

Supervisor: Prof. Dr. Judith Kelner

Co-Supervisor: Prof. Dr. Alejandro C. Frery Orgambide

Recife, September 2003.

To my parents Paulo Sérgio and Tereza, my brother
Marcelo, my sister Daniela and my love Paulyne.

ACKNOWLEDGMENTS

I would like to thank professor Judith and professor Alejandro for all the work, ideas and support in supervising this dissertation. I would also like to thank professor Djamel who helped me very much and brought many ideas and contributions to this work.

Thanks to professors Judith and Djamel for also allowing me to develop my work and building the necessary infrastructure in Recife.

To Kamienski and Stênio, many thanks to the support, the suggestions and for solving many doubts.

Many thanks to all the other people from GPRT (Networks and Telecommunications Research Group), who also offered support for this work.

Many thanks to my two great friends Ciro and Joaquim for also helping throughout the development of this work, especially helping me clear the mind through Age of Empires II and Counter-Strike matches.

I would also like to thank Paulyne, the woman of my life, for all the help, the love and the moral support to go on with this work.

Finally, I would like to thank my parents Paulo Sérgio and Tereza, my brother Marcelo, my sister Daniela, my grandparents Jairo (in memorian), Ivonilde, Messias (in memorian) and Arina, my uncles, my aunts and my cousins for all the faith and support to finish this work.

Contents

Figures Index

Tables Index

Abstract

The *Network Time Protocol* (NTP), a protocol that synchronizes clocks from networked computers, has nearly two decades of existence and is continuously evolving. Due to its robust treatment of phase and frequency, an algorithm that disciplines the local machine clock, implementations available for a great deal of platforms and operating systems, it became a "de facto" standard.

However, this protocol still has some drawbacks that need to be addressed to make it operationally more efficient, since its performance depends on good network conditions for a proper exchange of synchronization information and it suffers from any level of network congestion. Due to such limitations, in many countries (including Brazil) NTP is considered inadequate to provide time information in a trustable way, considering without legal value any time registries in systems that use it. Furthermore, many applications need a trustable clock control system to work correctly (for example, banking systems and geographically distributed database servers). This requires many companies to use traditional and more expensive legacy systems for a correct and legal functioning of their computer clocks.

With the use of Quality of Service in computer networks, this problem can be elegantly approached and solved. Many Quality of Service architectures were recently proposed, but the Differentiated Services architecture, due to its low implementation complexity and to the fact that it was the most studied and implemented architecture, is the strongest candidate to worldwide use in the coming years. This architecture fits the problem of clock synchronization very well, but the solution is not trivial, as is showing this work.

This dissertation suggests a framework to deal with clock synchronization using NTP in DiffServ domains with or without the use of bandwidth brokers. Using Well Defined Services (WDS) and based on the idea that applications should be aware of the type of treatment required by their traffic, this framework consists of the adoption of policies to treat NTP traffic at network nodes and the adoption of a policy to mark packets by the application. The proposal is validated with a case study performed with real measurement of the application performance over an environment based on network emulation.

Resumo

O *Network Time Protocol* (NTP) é um protocolo para sincronização de relógios de computadores em rede que tem quase duas décadas de existência e está em contínua evolução. Com algoritmos robustos para tratamento de fase e freqüência, com um bom disciplinador de relógio local e tendo implementações para as mais diversas plataformas e sistemas operacionais de computadores e equipamentos de rede, o NTP tornou-se um padrão de fato.

Entretanto, esse protocolo ainda tem problemas que precisam ser solucionados para ser operacionalmente eficaz, pois seu desempenho depende de boas condições de rede para a troca das informações de sincronização, sofrendo bastante em caso de congestionamento. Devido a esses problemas, em vários países (incluindo o Brasil) esse protocolo é considerado inadequado para prover informação de horário de forma confiável, fazendo com que registros de hora em sistemas que o utilizam não tenham valor legal. Além disso, muitas aplicações necessitam de um sistema de controle de relógios confiável para funcionar corretamente (por exemplo, sistemas bancários e servidores de bancos de dados distribuídos). Isso obriga muitas empresas a utilizar sistemas legados tradicionais e mais caros para poder funcionar de forma correta e legal.

Com o advento da Qualidade de Serviço em redes de computadores, esse problema pode ser abordado elegantemente e resolvido. Várias arquiteturas de Qualidade de Serviço foram propostas, mas a arquitetura de Serviços Diferenciados (DiffServ), devido a sua facilidade de implementação e ao fato de ter sido a mais estudada e implementada experimentalmente, mostrou-se a mais forte candidata à implantação mundial e no menor prazo. Essa arquitetura adequou-se bem ao problema de sincronização de relógios, embora a solução não seja trivial.

Essa dissertação sugere um arcabouço para lidar com a sincronização de relógios através do NTP em domínios DiffServ com ou sem corretores de banda. Tendo-se Serviços Bem Definidos (Well Defined Services – WDS) e baseado na idéia de que as aplicações devem conhecer o tipo de tratamento necessário ao seu tráfego, esse arcabouço consiste na adoção de políticas para o tratamento de tráfego NTP nos equipamentos de rede e na adoção de uma política para a marcação de pacotes por parte da aplicação. A proposta é validada com um estudo de caso feito com medição real do desempenho da aplicação sobre um ambiente de rede emulado.

Chapter 1 - Introduction

This chapter shows a brief introduction to the problems approached by this dissertation, justifying the importance of well-behaved clock synchronization for networked computers.

1.1. Motivation

Clock synchronization in computer networks is very important to the well functioning of most distributed computer systems and is fundamental to give integrity guarantees to some services.

The accuracy of clocks that are synchronized through computer networks depends on the delays to which the synchronization packets are submitted and especially on the variation of such delays (also known as *jitter*). Therefore, guaranteeing a minimum and stable delay to the transportation of these packets is a way of guaranteeing clock precision.

Though some statistical techniques are used to yield an acceptable solution for the problem, these could not deal with the totally unpredictable network behavior. Even today, there had been no proposal, to our knowledge, to improve the quality of clock synchronization based on network Quality of Service (QoS). With network QoS, it is possible to offer some guarantees on network delay and jitter to the synchronization mechanism, benefiting all the applications that depend on it.

1.2. Objectives

As a result of the observations of real-life systems that needed reliable time from clocks synchronized through IP networks, it was concluded that some of these clocks could not make the required guarantees. For example, it was common for some of the observed clocks that used NTP to lose synchronism and to drift too much from their reference time. In other words, this occurred due to network congestion (even on non-saturated links, due to bursts of traffic), which incurred into packet losses and high jitter. The network links of the observed machines were not saturated (full bandwidth utilization for long periods), but had occasional bursts of traffic that used the whole bandwidth of the link for short intervals, generating delay and jitter.

This work intended to make an analysis of the benefits quality of service may bring to clock synchronization, including the metering of quality parameters for the clock service provided to applications.

1.3. Related Work

Some more recent work [1] tried to improve the quality of synchronization by improving the algorithms for clock selection and time prediction based on local clock drift and network jitter measurements.

No proposal had been made to combine both the clock synchronization and Quality of Service architectures.

1.4. Organization of the Dissertation

This dissertation has been structured in order to introduce the concepts involved and present a satisfactory solution. It was written according to the norms of the Brazilian Association for Technical Norms (ABNT) [2]. The remainder of this document is organized as follows:

The second chapter describes the state of the art in clock synchronization, first detailing the problem, then explaining the basic concepts in clock synchronization and describing the evolution of the main solutions for clock synchronization for networked computers. It then explains the basic concepts in Quality of Service and describes the most important proposals for implementing it.

The third chapter examines the Network Time Protocol, which is the main protocol used for clock synchronization of networked computers and a "de facto" standard for clock synchronization.

The fourth chapter details the Differentiated Services architecture for providing Quality of Service on IP Packet Networks and the main issues related to its use.

The fifth chapter brings an improved description of the stochastic process of clock synchronization that helps better understanding of the problem of clock synchronization, makes a proposal for the problem of clock synchronization infrastructure on IP networks and shows how to implement it.

The sixth chapter explains a case study performed using network emulation to validate the proposal and details its results.

The seventh chapter draws a conclusion for this dissertation and shows some interesting future works to it.

All references used to develop this work are shown in the eighth chapter.

Finally, the appendix brings a list of abbreviations and acronyms and a glossary of terms that are used throughout the text.

Chapter 2 - State of the Art

This details the problem of making guarantees for clock synchronization. After that, it explains the basic concepts in clock synchronization and in quality of service, showing the most important known approaches for them.

2.1. The Problem

Even though the Network Time Protocol has evolved through many years of factual use, it is known that many time servers using NTP suffer from problems caused by bad network conditions. It is very common for an NTP machine to lose synchrony in cases of high network congestion and in cases of low network congestion if the machine is not dedicated to serving time. In less common but still recurring cases, some dedicated time servers lose synchronism even in situations of low network congestion.

This unreliability in time keeping makes NTP use not trustable for serving time to neither sensitive application nor to not-so-important but legally restricted logging applications. In some countries, Internet connection providers and Internet service providers must keep logs of user connection for months or even years and must present these logs in case the police or a court of justice requires so.

In the case of Brazil [3], for example, the providers are also responsible for the quality of the time used in these logs and federal government points at an official agency as the only trustable time supplier. This agency is responsible for serving time and enforcing secure and reliable time distribution methods. As expected, NTP is not considered a trustable time distribution method and it is only used to serve non-trustable time for user convenience reasons.

2.2. Basic Concepts in Clock Synchronization

The calendar system used in most of the world today is the **Gregorian Calendar**, created by Pope Gregory XIII in 1582, following the suggestions of the astronomers Christopher Clavius and Luigi Lilio. This system should be used for dates in the **Common Era** (AD, i.e., anno Domini) [4]. For a historical description of calendar systems, see [4].

In 1958, the standard second was defined as 1/31,556,925.9747 of the tropical year that began this century. But it was proved later that the duration of the year is not stable enough, relative to modern technology [4]. Given this instability, in 1967 the standard second was redefined as "9,129,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom". In 1972, the International Bureau of Weights and Measures (BIPM) defined the **International Atomic Time** (TAI) and started to operate – now the International Time Bureau (BIH) does – a cooperation among many observatories throughout the world to keep the **Coordinated Universal Time** (UTC), a standard that evolved from the Greenwich Mean Time (GMT). The hour is legally defined in most countries based on UTC.

Since the astronomical time (time of the day, phase of the moon, time of the year) doesn't evolve deterministically (stably), other standards were defined based on UTC. Based on apparent mean solar time, the **UT0** timescale was defined adding corrections for earth orbit and inclination over UTC, the **UT1** timescale (conventional civil time) was defined by also adding corrections for polar migration and the **UT2** timescale by also adding corrections for known periodic variations. Whenever the difference between TAI and UT1 approaches 0.7 second, a second - called **leap second** - is inserted or deleted from TAI in the last day of the month. For more detailed information on timekeeping standards, see [4].

In the design of a clock synchronization system for UTC, some important items must be considered. Trustable clocks (like GPS, that bring time and date information) and *time tickers* (such as atomic clocks, which show precisely the passage of time but must be adjusted to the correct time before starting to operating) are rather expensive. One of these must be connected to (or embedded in) a device that will serve timestamps to other devices (which can be a handheld computer, a military supercomputer, a microwave oven, a telephone exchange, etc) with less expensive and less trustable clocks. Such timeserving device is called a primary source of time. Since this device may fail, systems that require trustable time for proper operation need some redundancy (in equipment and/or networking) in time synchronization, making the system even more expensive.

When analyzing the quality of synchronization between two devices, an important metric is the *offset*, i.e., the difference, measured in seconds, between the times marked by both devices at a given instant. Some synchronization systems estimate this offset, or the maximum possible offset given the running conditions. Another metric is the *skew*, which consists of the variation of the offset with time and shows how stable is the system. Skew estimation is also important to notice if (and how much) the system is affected by different conditions like temperature changes or time of the year (season), if analyzed on long use of the system. The *drift* is an important metric that is defined as the variation of skew with time. It is useful for analyzing the stability of a system on long runs.

Since they are applied to the results of synchronization, these metrics can be used to analyze:

- The time distribution part of a time synchronization system with trustable clocks in both devices;
- A single local clock with a trustable clock as a reference and a trustable time synchronization system;
- The "additive" result of having a not-so-accurate clock and a time distribution system with a trustable clock as a reference (it can even show which part of the system is strongly affecting accuracy, if any: the local clock or the time distribution);
- Candidate reference clocks (comparatively) with a not-so-good local clock and a time distribution system. In this case, experience shows that with a few candidates maximum likelihood estimation gives very good tips on which candidates are trustable and which are not.

2.3. Clock Synchronization Systems Evolution

Many clock synchronization systems have been proposed in the last decades. In 1983 the **Daytime** [5] and **Time** [6] protocols were standardized. A Daytime server acts as a simple time supplier and offers time in human readable format. It is recommended that no machine should read this format to synchronize time. A Time server also acts as a time supplier, but it supplies GMT time (not local time) in a binary format for another machine (a Time client), which will simply update its own time with no further analysis of the received time. Both protocols have a resolution of 1 second, which is very bad for any serious time keeping needs.

The **Automated Computer Time Service** (ACTS) first went on-line in 1988 [7], operated by the National Institute of Standards and Technology (NIST). It is composed by timeservers and clients that communicate over telephone lines and modems. Since there normally are very few lines serving ACTS time in a country, it is common to use long distance calls, turning the system a bit expensive. ACTS timestamps have a resolution of 1 second, but it provides a propagation compensation scheme that guarantees very well configured systems with a dedicated server a maximum error of few milliseconds. It only provides phase correction of time oscillators, not clock discipline.

The **Distributed Time Service** (DTS) [8] proposed by the digital Equipment Corporation in 1989 is well suited for managed local network environments [4]. It exchanges timestamps between servers and "clerks" with a resolution of 100 nanoseconds. It provides error estimation and server selection algorithms, but similarly to the previous protocols it doesn't provide clock discipline. Just phase correction of time oscillators.

The **Fuzzball** routing protocol [9] incorporates time synchronization information directly into its routing messages, using minimum paths (one link) between the routers. It uses a spanning tree to propagate time offsets. This information is used to synchronize local clocks using the algorithm described in [4]. No server selection is used. It is designed for its local networks and doesn't apply to the Internet.

The **Network Time Protocol** (NTP) was first standardized in 1985 [10] and has much evolved since. It was initially based on the Fuzzball protocol, but after taking some ideas from DTS and improving many of its algorithms, the official implementation of the version 4 of the standard [11] is has proven to be the best technique achieved so far for time synchronization throughout the Internet. It is composed by a hierarchical set of time serving machines, where those attached to a high precision clock make the first level of the hierarchy. One of the main aspects of NTP is the introduction of clock discipline algorithms that not only adjust the phase of the local oscillators (clocks), but also their frequency. The timestamps in NTP have a resolution of about 232 picoseconds. A well-designed time system (with reasonably good local clocks and a predictable congestion-free network service) based on NTP can yield an error of about 1 millisecond. But almost all networks in the Internet have some level of congestion, which makes it hard to design a good timekeeping system in the Internet without the use of QoS.

2.4. Basic Concepts in Quality of Service

The Internet was first designed to be a military network where interoperability and resilience were the main issues. It should be able to recover quickly from problems (like a dead router) but wasn't supposed to deal with high congestion levels or service assurance.

With the growth of the Internet as a worldwide commercial network, many efforts were made to solve issues like accountability and predictable service. A few architectures were proposed for dealing with QoS. The three most important ones are presented in section 2.5.

2.4.1. Quality of Service Metrics

A service on the Internet can be defined by certain metrics required for its proper functioning. The most commonly used metrics are:

- Delay – the time a packet takes to go from the origin to the destination. It can be defined as maximum delay, as average delay or even as maximum delay for a percentage of the packets.
- Jitter – the variation of the delay. Some services (especially the ones with media streaming) are not much disturbed by delay, but suffer some quality loss with the variation of such delay.
- Bandwidth – the network bandwidth required by the service. It can be specified by minimum necessary, maximum that will be used (peak bandwidth) and/or average. Some common categories for defining bandwidth derive from QoS classes in ATM networks [12]: Constant Bit Rate (CBR), Variable Bit Rate (VBR), Available Bit Rate (ABR) and Unspecified Bit Rate (UBR).
- Loss – the loss rate that is acceptable by the service, generally calculated as a percentage of packets, or **Path Error Rate** (PER).

These are the most relevant metrics used for Quality of Service, but most proposals use only a subset of them and some do not use them explicitly, making a generic guarantee. The **IP Performance Metrics** (IPPM) Working Group from the Internet Engineering Task Force (IETF) [13] provided some discussions on these and some other metrics and how they can be measured [14][15][16].

2.4.2. The Best-Effort Service

The standard service offered by the Internet with no QoS support is known as Best-Effort (BE). It makes no metrics-based service guarantee; just the implicit guarantee that routers will do all they can to deliver the packets. Here, all packets get the same treatment, and packets carrying any type of data, from any connection, can be discarded due to congestion or physical layer (transport media) failure.

2.5. Quality of Service Architectures

Below are outlined the most important proposed architectures for dealing with QoS in the Internet and details of their behavior.

2.5.1. Integrated Services

Being the first QoS proposal made by the Network Working Group of the IETF [13], the Integrated Services (IntServ) [17] architecture was designed to be very effective at providing QoS for individual flows by reserving network resources on every router in a transmission path. The idea behind IntServ is that one cannot make guarantees on Quality of Service without making explicit resource reservations.

2.5.1.1. Main Service Definitions

Besides offering the Best-Effort Service, the IntServ architecture also offers two other types of service:

- Guaranteed QoS Service [18] – it provides firm guarantees on the throughput and delay of a connection, as long as the connection respects the agreed traffic profile. Such a connection can only be accepted after the verification that the network can implement it (admission control). This behavior is comparable to leased lines, and each connection needs some state information on each router. Therefore, it is impossible to scale IntServ to be deployed on the Internet, where some routers serve millions of connections simultaneously, making the processing of so much state information very expensive, if not impossible.
- Controlled Load Service [19] – it does not provide strong guarantees, packets will be treated as in a lightly loaded best-effort network. Most packets will be delivered successfully and will not be much more delayed than the minimum possible, as long as the submitted traffic is according to the agreed traffic profile. It was designed to support many applications being designed to the Internet today that don't need special guarantees but will not work on a heavily congested network. Each connection needs to use some state information on routers, as in the guaranteed service.

2.5.1.2. Architecture Components

IntServ is an implementation framework with four components:

- Signaling protocol – a protocol for reserving resources on the desired path. IntServ was designed to work in conjunction with any reservation protocol, but all experiments carried out used the Resource Reservation Protocol (RSVP) [20] for resource reservation, which was designed for Integrated Services. Therefore, the literature sometimes refers to the "IntServ/RSVP" architecture.

- Admission Control routine – it is responsible for verifying if there are enough resources for granting a requested service and should be used only during communication setup or reconfiguration.
- Packet Classifier – it classifies every incoming packet in a router to decide which service it should receive.
- Packet Scheduler – an algorithm used to make packets receive their requested QoS by scheduling them correctly.

2.5.1.3. The Resource Reservation Protocol

The Resource Reservation Protocol (RSVP) was designed for IntServ and is the only reservation protocol with a specification for it. RSVP is a receiver driven soft-state protocol, i.e., the reservations are requested by the receiver and they are valid for a limited time, being continuously refreshed until the end of the connection.

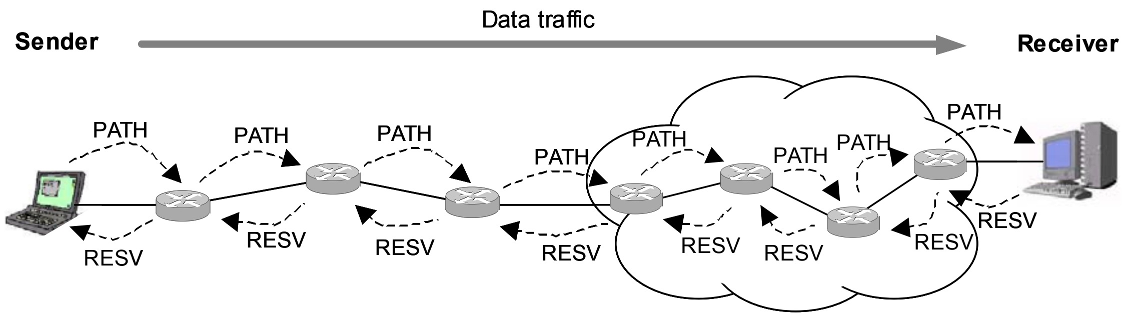

Figure 2-1: RSVP Message Exchange

The message exchange is done as shown in Figure 2-1 [32]. After the receiver requests a transmission from the sender, the latter sends back a PATH message, which will describe the required traffic profile and install some reverse routing information on each router along the way (chosen by whatever routing protocol that is in place) to the receiver. Upon receipt of this message, the receiver sends back a RESV message, which will traverse the reversed path (and not the path chosen by the routing protocol) reserving the resources in the routers. The RESV message is send hop-by-hop in order to follow the reversed path and make the correct reservations.

Considering that the reservation is soft-state and must be refreshed, the reservations must be refreshed often enough so that the loss if a single message will not falsely allow the removal of a connection [17].

2.5.2. Differentiated Services

The Differentiated Services architecture (DiffServ) [21] was proposed by the Network Working Group of the IETF four years after IntServ primarily to solve its scalability problems. Here, routers neither keep state information nor exchange signaling information. Therefore, the memory, processing and network control bandwidth requirements are much smaller and simpler to deal with.

The key idea in DiffServ is that services can be defined into classes, and all applications that require a certain service will be treated the same way. Packets are marked using a previously unused field (the **Type of Service** field – TOS [22] – now called **DiffServ Code Point** – DSCP [23]) in the IP header, which will not change protocol headers neither affect legacy systems that ignore the field. This treatment is called **Per Hop Behavior** (PHB).

2.5.2.1. Per Hop Behaviors

DiffServ has two official PHB specifications, besides the Best Effort:

- The **Assured Forwarding** (AF) PHB group [24] was defined to treat different types of traffic in relation to each other. There are four AF Classes, each with three drop precedences (12 combinations). It was designed for use of the same service within a class, where more important packets have a higher priority over others by being marked differently. The classes do not have precedence order between each other.
- The **Expedited Forwarding** (EF) PHB [23] should guarantee that packets in this class will not be dropped and their delay and jitter will be very close to minimum, if traffic agrees to the configured profile. It should function as a **virtual leased line** (VLL).

The Best Effort (BE) PHB, though not officially specified, represents the service already in place in the Internet and is sometimes called the **default** PHB. In DiffServ, no guarantees are made for BE, except that the BE will be made to forward packets correctly and that all packet flows of this PHB will be treated fairly among themselves.

2.5.2.2. DiffServ Network Topology

The DiffServ architecture has the following elements:

- The *DiffServ Domain* (DS Domain) is a network with DiffServ enabled.

- The *Core Router* (CR) is a DiffServ-aware router that can treat packets within PHBs according to the DSCP marking of the packet.
- The *Edge Router* (ER) has the same functions of the CR, plus *classification*, *marking*, *policy enforcement* and *traffic shaping*. It is supposed to be used at the borders of DS Domains to exchange traffic with other Domains (either DS Domains or not).
- The *Bandwidth Broker* (BB) makes contracts with end-users and with BBs from other DS Domains to request services. It is not mandatory for a DS Domain and has not been totally specified. There is no standard protocol for BBs in DiffServ to this date, just a suggestion that limits the possibilities in Diffserv to very few services [26].

2.5.2.3. Service Level Agreements

A Service Level Agreements (SLA) is a contract made between two networks with different owners. It can be physical (paper document) or virtual (digitally signed document using a BB).

A SLA must define network service guarantees, billing method, contingencies and penalties (in case of contract breach by any of the participants).

2.5.3. Multiprotocol Label Switching

While not being a real QoS architecture, Multiprocotol Label Switching (MPLS) is a packet-forwarding scheme that breaks and improves the hop-by-hop forwarding strategy used in the Internet, thus making a clear separation between routing and forwarding. It works between the Physical Media Dependent Layer (or Data Link Layer, in the OSI Reference Model [12]) and the Network layer (OSI and IP Reference Model [12]) by adding a header to packets and attributing a label on this header to allow forwarding.

MPLS uses Label Switched Paths (LSP), which gives a more effective control on where packets are being forwarded through. The MPLS enabled routers are called Label Switched Routers (LSR). LSRs keep some state information of forwarding paths, but no information on the passing flows, making them scale elegantly. The labels are defined on a per-link basis and can be grouped on Forwarding Equivalence Classes (FECs). Therefore, forwarding tables do not grow too much. New labels can be issued with a request based on traffic engineering or routing changes and can also be issued on demand due to a new path requested by a new connection.

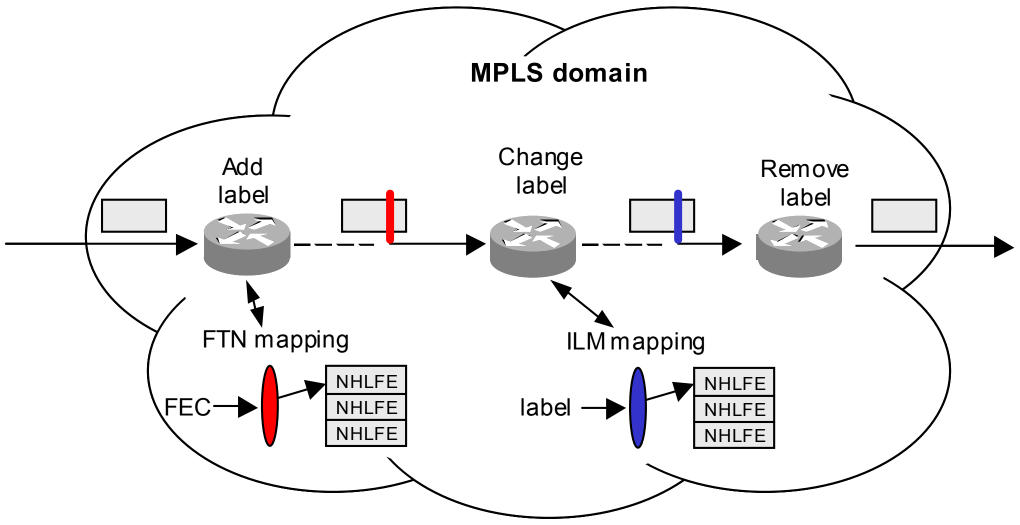

Figure 2-2: Packet forwarding and label switching in MPLS

The forwarding tables consist of Next-Hop Label Forwarding Entries (NHLFEs). These map incoming labels into outgoing interfaces and new labels, which are called Incoming Label Mapping (ILM). MPLS Domains interact with non-MPLS Domains through the use of FEC-to-NHLFE (FTM) maps to label unlabelled incoming packets and remove of unnecessary labels on border LSRs (see Figure 2-2).

2.6. Summary

NTP is the only reliable protocol for sharing time on top of IP networks. It has some problems with bad network connections and may require some network QoS to be properly deployed.

The IntServ architecture shows an elegant way of putting QoS on IP networks. Unfortunately, it is very costly and does not scale to today's Internet. MPLS is being used on many backbone providers, but its only effective on a big network with redundant paths. It won't of much help for users/networks with only one connection to the Internet.

The DiffServ architecture was chosen because of its actual deployment in some research networks worldwide and also because of its easiness of deployment and scalability. It is explained in more details in Chapter 4. In the next chapter we will discuss the NTP protocol and show some of its main weakness.

Chapter 3 - The Network Time Protocol

The *Network Time Protocol* (NTP) [1][11], standardized by the Internet Engineering Task Force (IETF), has been the "de facto" standard for clock synchronization for over 2 decades and has evolved with time. Nowadays, a huge (in hundreds of thousands) number of time servers [11][27] around the world utilize it to replicate time information to other machines.

3.1. NTP Hierarchy

NTP uses a hierarchical and distributed algorithm to share time, as shown in Figure 3-1 [4]. Physical clocks [28] are used as references of time and can be of many types: atomic, radio waves, crystal oscillators and GPS-based (Global Positioning System) clocks are the most common ones. Each of these clocks is attached to a computer called **primary reference** or **stratum 1** [11]. These computers serve a very trustable time, but aren't very common due to their high price, to the required skill to install and configure them and to the easiness of clock synchronization on networked computers.

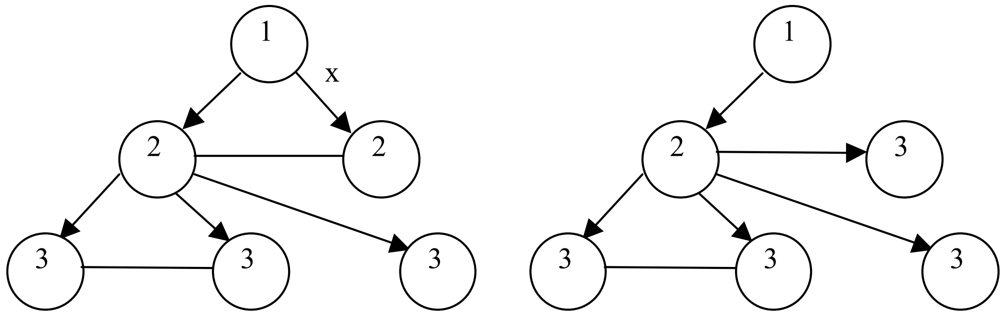

Figure 3-1: NTP Stratum tree (a) in normal hierarchy and (b) when link "x" fails

Any computer that synchronizes its time using a stratum 1 computer as a reference is then called **secondary reference** or **stratum 2**, and it may serve time to stratum 3 ones (also called secondary reference) and so on up to **stratum 15**, forming a synchronization tree. So, the stratum of a computer measures its distance to a physical clock, in network nodes. Each computer has an internal clock (normally a quartz oscillator) to keep time, called **local clock**, which is less trustable and very cheap. The idea is that the machines at the leaves of the tree do not need a very good precision. User machines are commonly stratum 4, but if you need a machine with very good time accuracy (banking or distributed database, for example) you should bring it upwards on the synchronization tree: directly connected to a physical clock (the machine is stratum 1 and NTP is only used to discipline the local clock) or synchronizing its time from a stratum 1 computer through a low jitter network path (in other words, one is better of buying a physical clock than paying the extra network link to one). A computer that is not synchronized to any other source is said to be stratum 16 (tough the protocol uses the value 0 as default [11]).

3.2. NTP Packet

Some information is needed for a correct packet exchange for clock synchronization. The NTP packet format [11] is specified in Table 3-1 with all the possible fields.

Table 3-1: NTP packet format

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
LI		VN			Mode			Stratum								Poll								Precision							
Root Delay (32 bits)																															
Root Dispersion (32 bits)																															
Reference Identifier (32 bits)																															
Reference Timestamp (64 bits)																															
Originate Timestamp (64 bits)																															
Receive Timestamp (64 bits)																															
Transmit Timestamp (64 bits)																															
Key Identifier (optional) (32 bits)																															
Message Digest (optional) (128 bits)																															

The **Leap Indicator** (LI) signals an impending leap second to be inserted or deleted in the last minute of the current day. This is indicated by astronomers as a correction of UTC time in relation to UT1 (see section 2.2) and can be manually set on stratum 1 servers or these servers must check periodically for the need of a leap second on an observatory server. So far, leap seconds have only been inserted on the last minute of the year and have never been deleted. The server sets this field.

The **Version Number** (VN) refers to the version of NTP being used. The latest version is 4 [11]. Both (the server and the client) set this field, but the server only replies to the request if it supports the version of the client.

Mode refers to the mode of synchronization, and can be: client, server, broadcast (poor synchronization) or symmetric (active or passive). The other 3 possible values are reserved. On symmetric modes the server maintains a state for each client, which is unreliable without access control (due to the unpredictable number of clients). Both the client and the server set this field.

Stratum is the stratum of the machine that sent the packet. Both the client and the server set this field.

Poll is the interval between two requests from a client, measured in seconds. It is always a power of 2. The client sets this field.

Precision is the precision of the local clock. Both the client and the server set this field.

Root Delay is an estimation of the round trip delay from client all the way to the root server (stratum 1), passing through the intermediary servers, in seconds (the first 16 bits represent the integer part). Both the client and the server set this field.

The **Root Dispersion** represents the estimated error from the root server. Both the client and the server set this field.

The **Reference Identifier** represents where the machine synchronizes. If the machine is a secondary reference, it must put the IP of its server. If it is a primary reference, these 4 bytes represent an ASCII string describing the local clock, e.g., GPS, NIST (modem service), WWV (radio waves), etc. Both the client and the server set this field.

The **Reference Timestamp** represents the time at which the local clock was last set or corrected. Both the client and the server set this field.

The **Originate Timestamp** is the time when the request departed from the client, in seconds (the first 32 bits represent the integer part). It has a reasonable 232 picoseconds precision, considering the guarantees of NTP in the range of a millisecond for well-configured time-keeping systems.

The **Receive Timestamp** represents the time the request arrived at the server.

Transmit Timestamp is the time when the reply departed the server.

The **Key Identifier** defines the authentication algorithm used. Both (the client and the server) if using any type of authentication set this field.

The **Message Digest** represents the checksum of all the fields in the packet, encrypted with the authentication key agreed by both the client and server. Keys are not exchanged through NTP. Both the client and the server set this field if authentication is used.

3.3. NTP Authentication

NTP can use authentication mechanisms to prevent having many unauthorized machines use its clock as reference to generate undesired network load that may compromise other clocks (including the reference clock itself). These mechanisms are beyond the scope of this dissertation and are explained in detail in [4].

3.4. NTP Metrics

Some interesting (and often confusing) metrics [4] can be used to analyze time synchronization problems and are explained here. They are estimated by the NTP implementation to help investigate the quality of synchronization of a given clock.

The *stability* of a clock is how well it can maintain a constant frequency, while *accuracy* is how well its frequency compares to time standards and the *precision* is how precisely these quantities can be maintained on a system (maximum error estimation). In NTP, timestamps (which measure seconds) are 64 bits long with 32 bits for the integer part (136 years) and 32 bits for the decimal part. This gives us a timestamp accuracy of 232 picoseconds.

The *offset* of two clocks (here, an NTP client and a server) is the time difference between them at a specific instant. The *skew* represents the frequency difference between them (computed as the first derivative of offset with time), measured in seconds/second or in parts per million (PPM) and the *drift* is the variation of skew (second derivative of offset with time). The drift is a good measure of stability.

Local quartz oscillator clocks have a high skew, and the clocks *drift* (i.e., the skew varies) too much. Frequent room temperature change is one of the most common causes of drift.

3.5. Clock Synchronization Process

The full stochastic process of clock synchronization is modeled in [1], but this work proposes a simpler one to study the clock synchronization, explained in section 5.1. In RFC 1305 [4], Mills shows a very simple description of this stochastic process, explained next.

The timestamp exchange is represented in Figure 3-2. In the i-th iteration, the client generates a timestamp T_{1i} and immediately sends a message to the server with this originate timestamp. The server receives the message and generates a receive timestamp, T_{2i}. After processing the message and generating a response, the server generates a transmit timestamp, T_{3i}, puts it on the response and sends it. Upon receipt of the message, the client generates another timestamp, T_{4i}. After this message exchange, the client has four timestamps, two of them generated by the server.

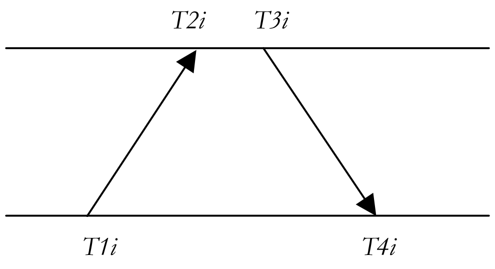

Figure 3-2: Timestamp exchange

Then, the client generates a sample of the clock offset δ and the roundtrip time θ estimations using the following equations:

$$\delta_i = (T_{4i} - T_{1i}) - (T_{3i} - T_{2i})$$

$$\theta_i = \frac{(T_{2i} - T_{1i}) + (T_{3i} - T_{4i})}{2}$$

NTP gives these computed values to the correction calculator, which will generate an estimation of time based on the last samples and give commands to the operating system to correct its time.

The interval between each *T1i*, also called *polling* interval (P), is based on how well the system is maintaining the synchronization. Generally, when NTP is started, it assumes the value of 64 seconds and rises (in powers of 2) up to the value 1024 (generating less network activity) if time keeping conditions are good. Noisy data (bad server or high network jitter) and many lost packets make this value shrink down to 64 seconds again.

3.6. Clock Filtering and Reference Clock Selection Algorithms

NTP client computers are normally configured with more than one reference clock (with possibly varying strata), but it utilizes some statistical techniques (specially maximum likelihood) to choose only the best ones. Some commonly used clock-selection algorithms can be found in [29][4] and are detailed below. The NTP Specification does not require these algorithms be implemented, but it requires that a secondary reference NTP server (a machine that synchronizes through NTP and serves time to others) uses at least algorithms with equivalent functionality.

At first, NTP polls all the candidate servers in order to put aside bad clocks (or clocks with bad internet connections to it), leaving only reasonably good clocks. For each polled clock, it calculates a dispersion value based on the total polling time, system precision and sample age. Then, it creates a table with the relative dispersion ε between each pair of server candidates (the dispersion of each one compared to the dispersion of all the others), weighted by the synchronization distance w (dispersion plus one-half the absolute delay) [4]. The equation to calculate the terms of the table is shown below:

$$\varepsilon_j = \sum_{i=0}^{n-1} \varepsilon_{ij} w^{i+1}$$

Through the maximum likelihood, it orders the clocks according to the growing mean relative dispersion.

3.6.1. Clock Filtering Algorithm

Clock filtering is used to select the best samples from a given clock. Each time a poll is accomplished, the sample is compared to other ones from the same server and all the samples are re-evaluated to discard bad and old ones.

Each sample generates three variables: offset, distance and dispersion. The dispersions of old samples are updated (since they depend on age) and samples with an offset or dispersion higher than the maximum acceptable (configurable) are discarded and will not be used for time correction calculation.

3.6.2. Clock Selection Algorithms

The clock selection is done using combinations of an intersection algorithm and a clustering algorithm. The intersection algorithm uses the offset and the dispersion of each clock to generate a clock interval for comparison. Some of these intervals will not have intersection, so an intersection interval of as many clocks as possible is generated in order to choose the best clocks and discard the others from the server candidate list. If it is not possible to choose an interval that reaches a simple majority of the clocks, no clocks will be used and synchronization will not occur.

The clustering algorithm orders the retained servers according to stratum and synchronization distance and similarly those with high distance can be discarded. The remaining servers will be polled regularly, but only the best ones will be used as a source for synchronization. Since this variable changes over time, the source can be changed if it becomes considerably worse than others, but in order to avoid instability it is done only when the difference may provide a very good deal of improvement in accuracy. Finally, the data coming form the selected clocks will be combined to produce a phase signal for the clock discipline algorithms explained next.

3.7. Clock Discipline Algorithms

In NTP, the clock discipline corrects the computer clock time, compensates for intrinsic frequency error and adjusts various parameters dynamically in response to measured network jitter and oscillator frequency stability (also called oscillator **wander**) [1]. This can be achieved through the combination of two feedback control systems. One is a Phase-Lock Loop (PLL), where updates are used directly to minimize phase error and indirectly to minimize frequency error. The other one is the Frequency-Lock Loop (FLL), where updates are used directly to minimize frequency error and indirectly to minimize phase error.

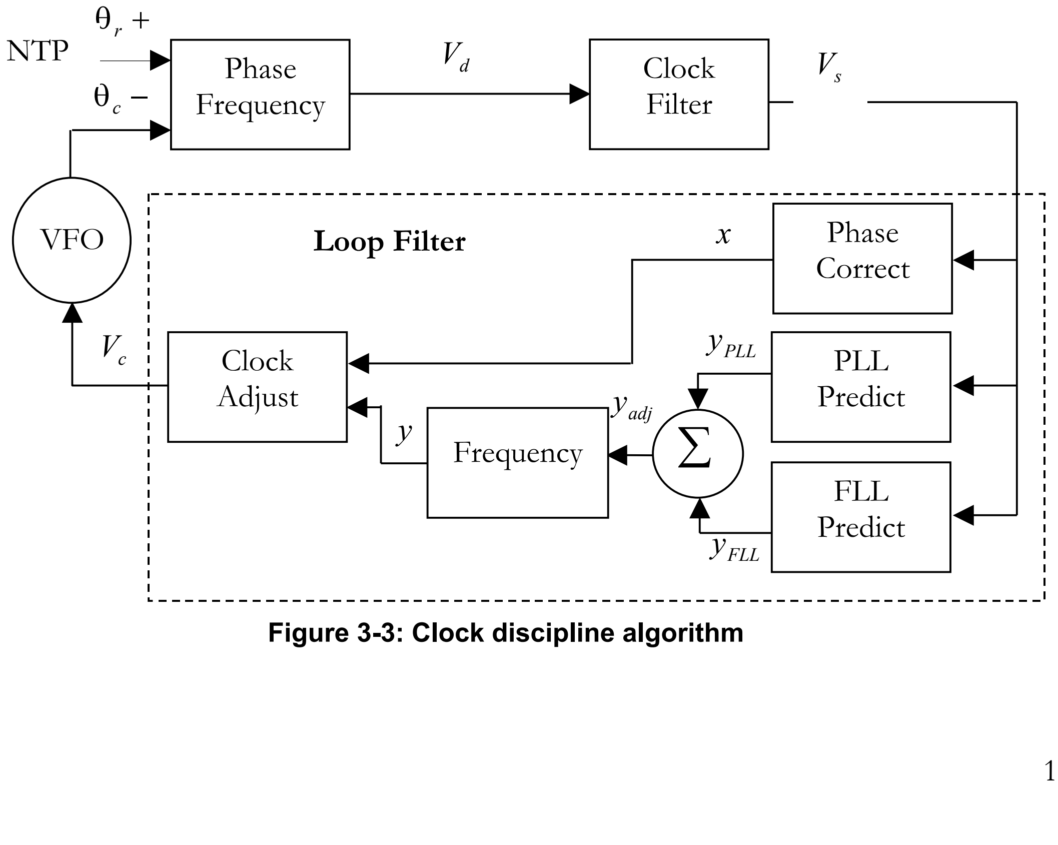

Figure 3-3: Clock discipline algorithm

The discipline is implemented as the feedback control system shown in Figure 3-3 [1]. θ_r represents the reference phase produced by the update (receipt of an NTP response) and θ_c represents the control phase produced by the variable-frequency oscillator (VFO), which controls the computer clock. The phase detector produces the instantaneous phase difference between them, V_d. The clock filter will filter, select and combine the best clocks to produce V_s. This sporadic signal will be converted in a periodic signal V_c (generated at intervals of one second) by the loop filter, which will use it to correct the frequency and phase of the VFO.

The loop filter uses a phase predictor (the PLL) and a frequency predictor (FLL). A hybrid PLL/FLL design (presently used in NTP) permits the loop filter to adapt to different running conditions. Depending on the polling period P, PLL or FLL has a greater weight on the resulting clock adjustments. The higher is P, the greater the weight of FLL. This allows for a big value of P with smooth (i.e., controlled) loss in accuracy.

The PLL loop predicts a frequency adjustment y_{PLL} as an integral of $V_s P$, using past (recent) values of V_s. This can drive both time error to zero with a fast convergence and can also drive frequency error to zero.

The FLL loop predicts a frequency adjustment y_{FLL} as an exponential average of past frequency changes, computed from V_s and P. The goal of the algorithm is to reduce future values of V_s to zero.

3.8. Quality of Synchronization

Mills says [28] that NTP yields better results (i.e., the offset can be measured with the smallest error) when the network load isn't high and even that is not a problem because a network link generally spends very little time with high loads. But practice shows that on these periods and even on some low load periods, due to the lack of control over packets bursts that generate jitter (because of queuing) and loss, NTP loses synchrony and the machine depends solely on its local clock to keep the time. Then, it must wait until the network "goes back to normal" and NTP [28] synchronizes again (which can take quite a few minutes).

Clock Synchronization depends on reference clock quality (which isn't much of a problem nowadays), local clock stability (though NTP tries to discipline the local clock), network delay and jitter [30]. Therefore, by guaranteeing minimum and stable network delay for NTP packets one guarantees permanently good synchronization for computers that use NTP to synchronize their clocks.

3.9. Summary

This chapter explained the main aspects of the NTP protocol, and detailed its packet format describing all fields, showed the main algorithms and how they guarantee a good functioning, described some metrics for the quality of synchronization and some possible problems when using NTP.

Chapter 4 - The Differentiated Services Architecture

DiffServ was designed to have the smoothest possible implementation in today's Internet by being able to work with legacy applications and network topologies [21]. Only router update is actually required, and it can even be done domain by domain, without impact to legacy applications.

The IPv4 header has a byte-long field named **Type of Service** (TOS), which was designed to help giving different treatment to packets marked by applications, specially for network controlling functions. The **IP Precedence** [22] used 3 of the 8 bits so that applications could signalize the need for low delay, high throughput and/or low loss. The other bits were ignored. In IPv6, a similar field, called **Traffic Class** [31], is defined in the header. The main idea in DiffServ is to map each configuration (a **DiffServ Code Point** – DSCP) of the new ***DS Field*** (which uses the first 6 bits of the TOS/Traffic Class and leaves the other two bits unused) to a different packet forwarding treatment, named **Per Hop Behavior** (PHB) [23]. Since the number of PHBs is limited by the size of the field, DiffServ routers treat only aggregated traffic flows, formed by a group of individual flows that will receive the same treatment.

4.1. Per Hop Behaviors

Some PHBs have been standardized, as explained below.

The **Best Effort** (BE) PHB, sometimes referred to as the **default** PHB, represents the treatment already used on the Internet. It supports fairness (among packets this same class) without service guarantees. Some minimum bandwidth can be reserved, but no guarantees are made, except the guarantee of fairness among packets. It was not formally standardized within DiffServ. The old definition from [22] is still current.

Assured Forwarding (AF) [24] is a PHB Group that offers some priority discard guarantees, especially among flows of the same AF class. There are 4 AF classes defined, each with 3 different drop precedence values, making a total of 12 different DSCPs. AF classes may be used by an application that uses different traffic flows, where some flows are more important than others (for example, layered video transmission) or by some single-flow applications with differentiated importance. Twelve codepoints are recommended for use within the AF PHB.

Expedited Forwarding (EF) [25] guarantees that the delay and the jitter of the packets will be very close to the minimum possible. EF traffic rate should be independent of other traffic passing on the same router. From the application's perspective, it should work as a virtual leased line (VLL). The recommended codepoint for EF is 101110 (in binary).

4.2. DiffServ Network Topology

The DiffServ network topology has many elements, as follows.

A **DS Domain** is any network structure that can treat DiffServ traffic. It can also interact with other non-DS Domains, but then no guarantees can be made.

The **Core Router** (CR) is a DiffServ-aware router that can treat packets within PHBs according to the DSCP marking of the packet. They are supposed to be used within DS Domains, but not on their borders (without connections to routers in different DS Domains).

The **Edge Router** (ER) is a router that performs the same function of the CR as well as **packet classification**, **marking**, **policy enforcement** and **traffic shaping**. It can be located at the border of a DiffServ Domain or inside the Domain, connecting end-users or customer networks. The packet classification can be based on many fields of the headers of the Network and Transport layers (of the IP Reference Model [12]) and should define a PHB to treat each incoming traffic flow. The policy enforcement checks if user traffic is according to the **Service Level Agreements** (SLAs – see section 4.3 below). The traffic shaping can be used to turn misbehaved traffic into well-behaved one (e.g., buffering bursty traffic and sending it at a constant rate). It does all this work according to the SLAs it is subject to. Since there are often many things to do, ERs are bottleneck potentials [32] of a DiffServ domain and must be very well provisioned. Therefore, they could be a lot more expensive. If packet marking is trusted to the applications that create them, there's no need for packet marking at the ER connected to customer networks. But since any network can use their own private markings (besides the standard PHBs), some translation (i.e., remarking) must be done at the ERs to accept traffic from other DS Domains.

The **Bandwidth Broker** (BB) [21] makes SLAs with network entities automatically to request services on-demand according to specified requirements (bandwidth, maximum delay/jitter, maximum drop probability, etc.) and after an agreement it informs the affected routers of its domain of the necessary changes. There should be one BB per domain and it should be accessed by other DS Domains (BBs calling each other) or by customer applications. Though proposed as part of the DiffServ architecture, BBs have no full specification whatsoever and are not used in DiffServ networks to this date. A suggestion for the restricted use of BBs can be found in [26].

4.3. Service Level Agreements

The DiffServ architecture doesn't define a protocol for the applications to reserve any resource in DiffServ routers. The customer must have a **Service Level Agreement** (SLA) with its provider to specify what services are required, what guarantees will be offered by both participants, how the service will be billed and what will be the contingencies and penalties for not following the agreement - which can range from dropping more packets than was agreed (provider fault) to putting more EF traffic on the network than was agreed (customer fault). SLAs are also made between providers, or DS domains, as shown in Figure 4-1.

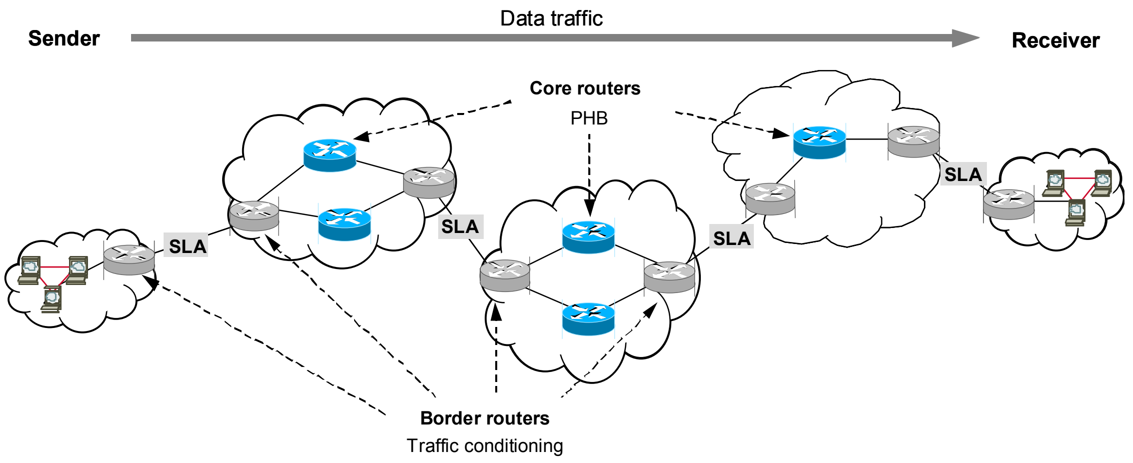

Figure 4-1: DiffServ logical architecture

Due to the aggregated treatment of traffic, DiffServ routers cannot make guarantees on microflows (e.g., a drop guarantee of less than 1% for EF traffic doesn't mean that every application that makes use of EF will experience less than 1% packet drops). It just guarantees that the average will be respected.

SLAs can be static (described by a document signed between a provider and a customer or between two providers) or dynamic, through the use of BBs (digitally signed), as shown in Figure 4-2 [32].

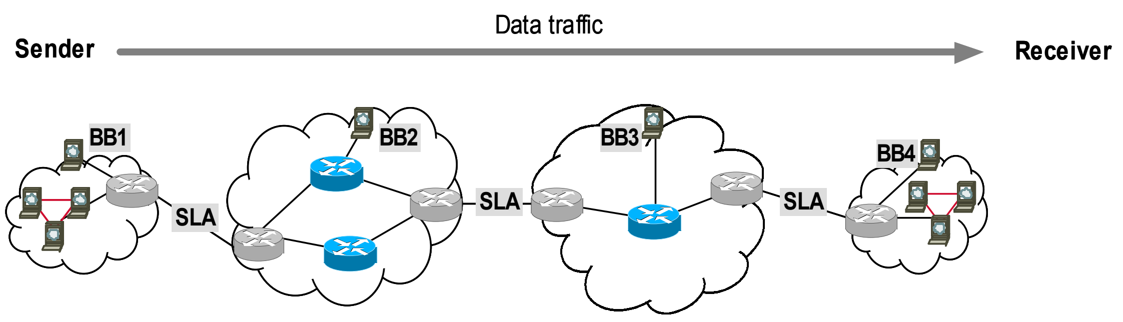

Figure 4-2: DiffServ delivery process with bandwidth brokerage

4.4. Bandwidth Brokerage

The use of Bandwidth Brokers is a tendency as well as a requirement for the acceptable deployment of a DiffServ network. When analyzing the time-scale for network provisioning (Figure 4-3) [33], one realizes that waiting for people to sign a document (even if a digital document) is not a reliable way of doing provisioning and not the best way for capacity planning. The costs of installing a better communication infrastructure than what is needed at the moment will be surpassed in the long term by the costs of updating that infrastructure many times.

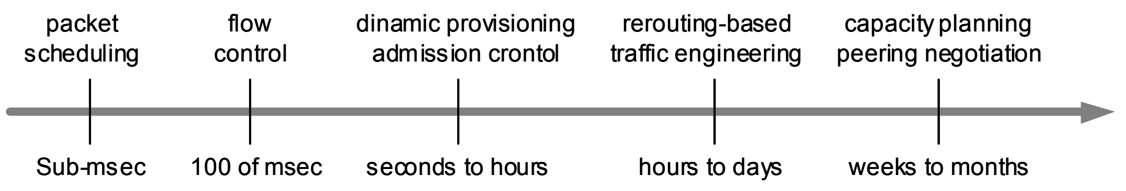
packet scheduling
flow control
dinamic provisioning admission crontol
rerouting-based traffic engineering
capacity planning peering negotiation
Sub-msec
100 of msec
seconds to hours
hours to days
weeks to months

Figure 4-3: Network provisioning time-scale

Despite this tendency, there is only a specification of a few parts of a BB [26], with restricted application.

4.5. Summary

This chapter presented the Differentiated Services architecture and how it treats aggregated traffic flows, making it scale well for higher usage necessities. It also detailed the elements of the DiffServ network topology and the necessity and use of Bandwidth Brokers to dynamically implement alterations in Service Level Agreements.

Chapter 5 - Proposed Alterations for Clock Synchronization Infrastructure

This chapter presents a novel networked clock synchronization stochastic process modeling for understanding the behavior of the NTP, proposes some alterations in the network infrastructure for applications that have low-delay and low-jitter requirements and provides some recommended configurations for a proper functioning of NTP with respect to the network infrastructure alterations. The novel stochastic process modeling is described in section 5.1. The detailed proposals are presented in sections 5.2 and 5.3. These proposals result in better and trustable clock synchronization for networked computers, as will be seen in section 6.3.

Though NTP is a connectionless protocol using UDP, we will use the term connection to refer to NTP flows (i.e., a conversation between a single client and a single server).

5.1. Stochastic Process Model

A stochastic process modeling of NTP is proposed to help understanding the clock synchronization process. This stochastic process is simpler (less detailed) than the one presented in [1], though more expressive for the intended approach and applications.

This new description aims to explain only the relationship of NTP between a client and a server, not trying to explain the relationship between two NTP machines in symmetric mode neither trying to describe the computations inside the NTP client. Security issues involving keyed authentication have not been considered either. No analytical considerations are made on this model.

A graphical explanation of the model will be used to enhance the description (as shown in Figure 5-1).

The client machine has a local time t_ℓ and tries to make this time be as near as possible to a reference time t_r. Local time does not evolve in a linear way in relation to reference time. Local clocks generally have a bad precision and their time varies (skews) from reference time in a non-linear way, causing drift. This characteristic makes it very important for a machine to periodically poll a reference machine for a trustable time, in order to adjust (diminish offset from server) and discipline (diminish frequency error) the local clock. Below are described the order of events for one iteration.

At time t_i, a time-order o_i leaves the client towards the server. The server receives this order at time $sr_i > t_i$ and, after a little computation (mainly packet processing and client information sanity checks), it sends back an answer at time $a_i > sr_i > t_i$. This answer arrives at the client at time $cr_i > a_i$. The times o_i and cr_i are measured with respect to the local time, and the times sr_i and a_i are set according to the time of the reference machine (in the following "reference time" for short). The next time-order o_{i+1} is sent a certain period after the order o_i, according to characteristics of the local clock and the network service received. This period is limited by *minpoll* and *maxpoll* configurations of the NTP client. The most common and accepted configuration of period P is the default:

$$P = 2^x \text{seconds} \mid 6 \le \text{x} \le 10$$

Therefore, the minimum value of P is 64, and its maximum value is 1024. The transition between different polling periods is always by changing between adjacent values of x. Whenever a frequency change is to occur, NTP analyses if this change is caused mostly due to local clock instability or due to network instability (jitter). This can be done analyzing the recent samples of packet delays (round trip time) to calculate jitter. In case of local clock instability, the polling period is increased (i.e., the polling rate diminishes). Otherwise, it is decreased.

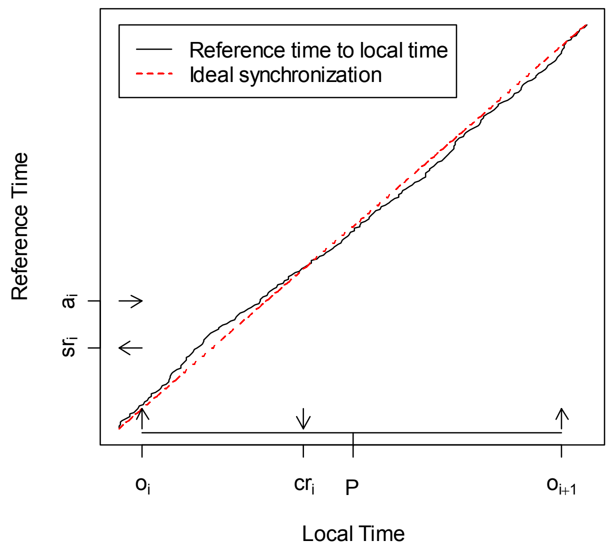

Figure 5-1: Clock synchronization stochastic process illustration

Figure 5-1 shows in solid line the relationship between time marked by the local clock (abscissas) and the "real time" provided by the reference machine (ordinates). The ideal synchronization, i.e., an identity relationship, is shown as reference in dashes. The relevant events are highlighted with arrows: o_i, cr_i, P and o_{i+1} along the local time axis, and sr_i and a_i along the reference time axis. Pointing upwards (o_i and o_{i+1}) and to the right (a_i) arrows denote outgoing events, while downwards (cr_i) and to the left (sr_i) arrows denote incoming events. The period of time between time orders P is also shown in the figure. The goal of a clock synchronization system is to decrease as much as possible the area between the lines and to define hard limits for the maximum possible difference between the curves at a given moment. The vertical distance between the lines represents the offset between the clocks, and diminishing this distance at all time is a way to diminish the area between the lines.

5.1.1. Association State Machine

The complete specification of the assumed stochastic model consists of providing laws for the time between the relevant events, along with their correlation structure (or lack of correlation structure). This will be done in the following.

The probability of events on an NTP association is explained in the state machine shown in Figure 5-2.

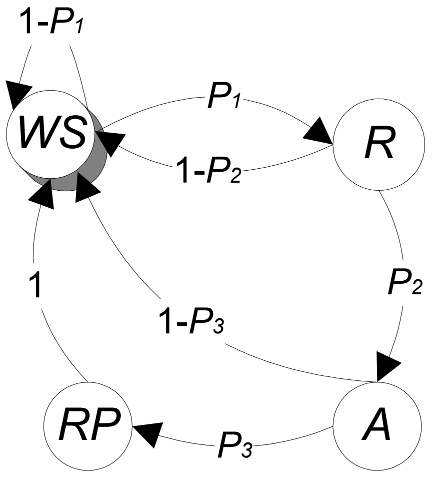

Figure 5-2: NTP association state machine

Each transition has a probability of occurrence. Assume the initial state is "Waiting to Send" (*WS*). The first transition, from state *Waiting-to-Send* (*WS*) to *Receiving* (*R*), represents the request packets leaving the client, and P_1 is the probability that this request reaches the server. If it doesn't, it means the network has discarded the packet, event that occurs with probability $1-P_1$. The second transition, from R to *Answering* (*A*), represents the acceptance and the processing of the request and occurs with probability P_2, i.e., with probability $1-P_2$ the server does not accept the connection (e.g., an unauthorized user tries to contact a server that used authentication). The third transition, from *A* to *Receiving-and-Processing* (*RP*), represents the event that the packet sent by the server reaches the client, and it has probability P_3. With probability $1-P_3$, therefore, the returning packet will be discarded by the network. The fourth transition, from *RP* to *WS*, will always happen if *RP* is reached.

The times taken by the three latter states, by the fourth transition and by the error transitions is ignored. The times taken by the first, the second, the third transition and by the *WS* state, are represented, respectively, by t_1, t_2, t_3 and t_4, where $t_1 + t_2 + t_3 + t_4 = P$, which is the period between requests. Whenever a packet is lost the respective poll will have no effect on the clock discipline, i.e., a packet loss doesn't incur in wrong data being processed by the client.

5.2. DiffServ Configuration

The Differentiated Services architecture was chosen for this work because its been used on many research networks and also because of its good scalability, as explained in section 2.5.

In a network that will serve NTP traffic, this work recommends the use of one of two possibilities: the Expedited Forwarding PHB or the Hot-Potato Forwarding PHB.

5.2.1. The Expedited Forwarding PHB

The EF PHB can be used to deliver NTP packets. Due to its efforts for providing minimum delay and no loss NTP would have a good stability with such a treatment. But NTP doesn't need the loss guarantees of EF and the delay guarantees could be a little improved. Even though the guaranteed bandwidth of outgoing EF connections should be big enough to accommodate all the incoming EF traffic [25], due to simultaneous arrivals and small bursts it is common to have a little queuing of EF traffic. This queuing can hurt synchronization quality.

In order to configure EF traffic on the routers of a DiffServ network that will transport NTP traffic, there should be a good estimation of how much NTP traffic will be served by this network (or the exact number of connections, if the network uses a Bandwidth Broker). There should also be an estimation of the maximum number of simultaneous packets, so that routers won't have to discard packets, which could hurt EF traffic other than NTP.

The EF configuration proposed **must** be done by a system administrator (either a person or a software, e.g., a BB) by adding to the reserved EF bandwidth of a link the bandwidth that will be used by NTP. As seen in section 5.1, this necessary bandwidth will depend on the configurations of NTP, where the standard behavior is to send and receive a 76-byte packet (or a 96-byte packet with authentication) every 1024 seconds, giving approximately 0.6 bps (or 0.75 bps with authentication) per NTP connection. Though the average bandwidth for NTP can be obtained by adding the bandwidth of each connection, it will not be equal to the peak bandwidth used by NTP traffic, since the requests will not arrive uniformly sparse. However, practice shows that on considerably loaded NTP servers traffic peaks are rarely more than twice the average. Therefore, system administrators **must** reserve twice the average bandwidth for NTP Traffic and, as specified in [25], if the way the EF PHB is implemented at a router can starve other traffic, the router **must** also use a token bucket (or a functionally equivalent mechanism) to limit the rate of EF at the reserved bandwidth.

In case the DS Domain uses a Bandwidth Broker and applications and foreign brokers can make requests to the broker, the reserved NTP traffic matrix will be known for each interface at each router of the domain. If the DS Domain does not have a broker, periodic measurements on Edge Routers (connected to other DS Domains or to customer networks) can show the necessity of NTP traffic and may be used to estimate future traffic necessities. With this information, the appropriate configurations for EF traffic must be done on Core routers and on Edge routers (here, with policing) of the DS Domain.

As stated in [25], EF packets carrying NTP traffic can have their DSCP remarked when entering another domain to a different codepoint, as long as the marking is treated in that domain by a PHB that satisfies the EF PHB specification. In case of tunneling of EF packets with NTP traffic, the encapsulating packet must also be marked as EF.

5.2.2. The Hot-Potato Forwarding PHB

This work introduces the Hot-Potato Forwarding (HPF) PHB, which is intended to be the ideal PHB to treat packets with strong low delay requirements and without delivery or bandwidth requirements.

The Hot-Potato Forwarding PHB can be used to construct a lossy, very low latency, very low jitter service through DS Domains. It's like a person receiving a hot potato on his or her hands: in order not to get the hands burned, he or she cannot keep holding it and should immediately send it to the next person. If one can't send the hot potato to the next person immediately, he or she should simply drop it.

Some services require very low delay and minimum jitter, but don't require delivery guarantees. Applications that need to send time-sensitive information, like the exchange of high-resolution timestamps (NTP) or the monitoring/metering of physical links delay (with the SNMP [34] philosophy that monitoring should not hurt operation and losses are acceptable) can make good use of such a service. HPF suits these applications better than EF due to the limited predictability of the delays in the latter. And with the loose bandwidth requirements of HPF, it will not disturb EF traffic.

HPF can be achieved by configuring a mechanism that will not delay traffic (e.g., a queue that only accepts one packet) and that will discard any packet that cannot be immediately sent (otherwise it would affect other traffic). DS Domain Edge routers should also police incoming HPF traffic to prevent it from flooding a network. HPF PHB will not use any traffic shaping that increases the delay of packets.

It is not recommended that a DiffServ router to serve a significant amount of HPF traffic. The general idea is that HPF traffic should not utilize more than 0.1% of a link, except on very special cases (e.g., a link to a public time server or an administrator-driven metering of a possibly problematic link). And since EF traffic has very restricted rules for accomplishing its guarantees, the implementation of HPF must always be done with care, so that it won't break the guarantees of EF.

The HPF PHB is not an obligatory requirement of any DiffServ network and is intended for private use within a DS Domain or between DS Domains that agree on its usage. If HPF is ever specified as a standard PHB, its implementation will never be mandatory to any network in order for it to be considered DiffServ compliant, as stated in [21][23].

5.2.2.1. Definition of the HPF PHB

The HPF PHB provides forwarding treatment for a particular DiffServ aggregate called HPF traffic. HPF packets arriving at any Diffserv router **should** be immediately forwarded or dropped. No HPF packet should wait longer than a packet time (the time to send a packet in the outgoing network interface) to be sent. This assures a limited minimum jitter, based on the number of nodes and the packet serialization delays at each node.

No minimum bandwidth reservations are necessary or even desired. But a maximum bandwidth is important so that misuse of HPF will not affect other PHBs, notably the EF PHB. The HPF traffic **should not** break guarantees of other PHBs. The decision of discarding or scheduling an HPF packet must be done based on the guarantees of other traffic. In the event where the scheduling of an HPF packet results in the breaking of the guarantees of other traffic this HPF packet should be discarded. A DiffServ router that implements the HPF PHB **should not** use traffic shaping on HPF traffic other than the discard of nonconforming packets.

5.2.2.2. Implementation of the HPF PHB

Some types of scheduling mechanisms can be used to deliver a forwarding behavior similar to the one described in the previous section. But only two mechanisms have shown to be capable of providing the necessary means to accomplish all the required guarantees within a proper DS Domain: **priority queue** (PQ) and **class based queuing** (CBQ).

With PQ, the HPF traffic can be served by a separate queue (for HPF traffic) with the highest priority among all queues. If the queue size is measured in packets (it may differ among different router), this queue should have the size of one packet. If the queue size is measured in bytes, it should have the size of the biggest packet that it is supposed to accept. In order to prevent HPF to starve EF traffic, a token bucket **must** be used to limit HPF rate and two consecutive HPF packets should never be sent if there's an EF packet waiting.

Using CBQ, there are many possibilities of configuration. A recommended one is to use a queue for HPF (with a rate limiting mechanism to discard out-of-profile traffic, e.g., a token bucket) and a queue for EF, where the HPF queue will have a higher priority and be sized to a packet (if the queue is measured in packets) or to the size of the biggest packet expected (if the queue is measured in bytes). Again, if an HPF packet arrives when another one is being sent and there's an EF packet waiting, the HPF should be discarded. Other recommended possible implementation is to use a high priority class (no other class should have a higher priority) with two subclasses controlled using **weighted round robin** (WRR): one for HPF (single-packet queue) and another for EF, where both HPF and EF have the same weight and a maximum transmission of one packet per turn (to avoid any queue to wait longer than a packet to transmit). This WRR must be non-work conserving, i.e., if a queue has no packets on its slot, the other queue can transmit.

In the event where the HPF PHB is used on a DS Domain, the **recommended** codepoint for HPF is 010011 (binary), which is in a **local use** DSCP range (as defined in [23]). Here, as suggested in [23], no efforts are made towards compatibility with legacy (now mostly unused) low-delay TOS field markings [22], except the recommended [23] compatibility to the IP Precedence sub-field [22], where 010xxx represents immediate service.

5.2.2.3. HPF PHB Cautions

If two DS Domains exchange HPF traffic, they should agree on the DSCP for HPF or should do packet remarking (as long as the new marking are mapped into an equivalent PHB, according to the HPF definition above). If HPF packets are tunneled, the encapsulating packet must be also marked with the same DSCP.

HPF can be employed on any DiffServ router that implements any other PHBs, as long as the HPF specification is respected.

Even though no guarantees on the reliability of HPF traffic are made, a router should only discard an HPF packet if it really has to, in order to avoid starvation of HPF traffic. Though no bandwidth is reserved for HPF, the fact that no other priority should be higher than the one of HPF may make it interfere with the jitter observed by other traffic, especially BE.

5.3. Network Time Protocol Configuration

Some configurations are recommended for NTP in order to achieve a good and stable synchronization with distant servers on the Internet, according to the infrastructure presented in the previous section.

Some relevant issues must be discussed to gain insight into the aspects that influence clock synchronization. Applications like banking and geographically distributed databases (which includes simple logging of activities on many machines far away from each other) have very high time precision requirements and the configuration of their synchronization infrastructure must attain to each detail.

Any serious time-keeping purposes will require local-network dedicated time servers (running NTP) that will synchronize from remote computers and serve time to local computers. Using a computer for other tasks besides time keeping affect significantly the quality of the time served. Since NTP was designed for multitasking processors, giving it a higher priority can improve the efficiency of the clock discipline algorithms and even lower the jitter of network traffic (since the arrival and treatment of packets will preempt other tasks). Here, equipment that introduces considerable network jitter (e.g., an Ethernet Hub or a radio antenna with frequent collisions and retransmissions) should be avoided where possible, especially between the local time servers and the remote ones (where the presence of many intermediary nodes raises the values of the delay and the jitter).

Special care should be taken to use servers whose path to the client shares as little as possible Internet links, e.g., it is not recommended to use three servers located far away and all three on the same network, because if one of the intermediary communicating links fails then all servers will be unreachable.

5.3.1. Using the Expedited Forwarding PHB

Using too many servers to synchronize to is not a good option. NTP will always synchronize to only a few of the configured servers and will only switch servers if the quality of the data coming from the selected server gets worse than the others (which is not common). Furthermore, even though a few servers will be used, only up to 10 servers will be polled (including the chosen ones) to compare their times and switch among them when necessary. This way, each NTP machine will spend much more bandwidth then it was supposed to. When using EF PHB for NTP traffic, twice the average bandwidth will be reserved and packets will never be discarded, decreasing the available bandwidth for other applications.

Therefore, this work **recommends** the configuration of up to 3 servers for an NTP local server using the EF PHB. Only one good server is necessary on normal operation, but this redundancy is necessary for safety reasons. If the network has more than one connection to the Internet or to the local network (two or more network interfaces on the machine), at least one of the three NTP connections should use it.

The use of three servers will reserve about 6 times the bandwidth of a connection on each link (since links will differ for each remote server, it will actually be twice the bandwidth of a connection for the path to each remote server). It will make a bandwidth reservation of 3.6 bps for EF traffic, being 1.2 bps for each remote server.

This reservation on all the routers along the way must be done by the network administrator (estimation) or by a BB (exact). Either case, the packets should be marked by the application or by an Edge Router outside the customer network (which should be the first router connected to the customer network).

5.3.2. Using the Hot-Potato Forwarding PHB

Though no bandwidth is reserved with the use of the HPF PHB, the same restrictions on the number of servers apply here. Using more than three servers is unnecessary. This work **recommends** the configuration of 3 servers for an NTP local server using the HPF PHB, but no less than 3 servers. Even though NTP behaves nicely in the face of high packet loss rates and the fact that HPF packets will improve the quality of the synchronization to its best (due to the assured low jitter), the remote possibility that the machine will stay hours without hearing from any server should be avoided. Here, it is also recommended that servers be chosen carefully, preferably from different nearby (i.e., with low delay to reach) networks.

An estimate of the maximum use of bandwidth is necessary in case a network manager configures the routers along the paths of HPF traffic. If a BB is used, this information will be accurate enough and supplied by the application to the BB. Packet markings to the correct value corresponding to the HPF PHB must be done by the application itself or by the Edge Router connected to the customer network.

5.4. Summary

This chapter explained how the NTP machines interact, showing a novel model of stochastic process for clock synchronization and a state machine for an NTP client/server association, detailing the events and their relation.

This chapter also proposed changes in the network infrastructure based on the DiffServ architecture to allow for a good quality of synchronization for clocks on networked computers. It recommended the use of the Expedited Forwarding PHB or the newly proposed Hot-Potato Forwarding PHB, which assures low and limited jitter. The theory and implementation of the HPF PHB were also specified.

Some NTP configuration recommendations for proper quality of synchronization were explained both for use with both the EF and the HPF PHBs.

Chapter 6 - Case Study with Network Emulation

This chapter presents a case study of the alterations introduced for the clock synchronization infrastructure presented in the previous chapter. The paradigm of network emulation was chosen for its evaluation, as opposed to network simulation and live network measurements, for reasons further explained. This allowed us to have more control over the experiments and helped bring results much closer to reality.

6.1. Introduction To Network Emulation

There were three possible ways to perform the case study of the proposed alterations for clock synchronization infrastructure: simulation, emulation and live network tests.

Using simulation would be cheap and runs faster than real traffic. The tests would be controlled and reproducible (which is necessary for making statistical analysis). The time a simulation runs can be faster than a live test by several orders of magnitude. Given the abstraction in a simulation, implementing simulation code would be much faster than implementing a complete code for a live test.

The measurement of live traffic in real networks would yield far more trustable results and reuse of the already deployed code (with very simple changes), but the results would not be reproducible (because of the environment not being controlled) and their analysis would be much more difficult (without statistical confidence). Live tests need to run in real-time. In other words, it can take very long time to make a full test. The necessary infrastructure to such tests would have prohibitive costs.

Using network emulation allows one to run "real" code (i.e., code used on production environments or intended for it) on a controllable and reproducible environment [35], hence the tests can interact with live environments as much as possible, cheap or good to analyze. Network devices and applications see the emulated network as a regular network and cannot tell the difference. A good point in favor of the emulation is this selective abstraction. One can choose which parts of a test could be emulated and which ones should be implemented, based on the test requirements, the present (or possible) infrastructure and the time available to implement more parts of the infrastructure. If the code that has to be implemented from scratch, emulation has the same problem of live tests: it will take longer to start the tests because of the code implementation. Using network emulation there's also the problem of running the tests in real-time.

A simulator for NTP is available [1] and is distributed along the source code of the standard implementation of NTP (publicly available). This implementation has no network standards implemented whatsoever and simulates only the core algorithms of NTP fed with real data traces. This code does not interact with any present network simulator. Re-implementing it in a network simulator (or even re-implementing all the necessary network standards to work with it) would consume too much time and would also run the risk of becoming a poor representation of the real environment (as any simulation effort that is not compared to a real implementation). The studied applications and almost all the necessary traffic controlling mechanisms had already been implemented. The chosen method for this work was network emulation.

6.2. Testing Environment and Experiment Configurations

Many elements were needed to build the necessary mechanisms to run the tests. It was important to duplicate the real Internet conditions as much as possible, and the following software tools were used for this purpose:

- A recent and stable implementation of NTP (as detailed in Chapter 3). This work used version 4.1.1b (from October 2002) of the main implementation of NTP (which can be found in [36]) with most of the configurations set as the default.
- An operating system that would support DiffServ, network emulation, traffic generation, NTP and that would allow configuring all of it. Due to past experience, **Linux** with kernel 2.4.18 was chosen and used on all machines except the stratum 2 time server (which used **FreeBSD**) and the stratum 1 time server[1] (Linux with kernel 2.4.2). The kernel was recompiled to allow for a network emulation tool and advanced router capabilities.
- A tool to configure the advanced router capabilities required by DiffServ [37]. **Traffic Control**, a utility of the **iproute2** package [38], was used for this. The "ping" tool, available in nearly any Internet host, was also used to measure jitter.

[1] This machine is located in the Brazilian National Observatory, which is responsible for the Brazilian legal time (HLB – *Hora Legal Brasileira*).

- A network emulation tool. After evaluating **DummyNet [39]**, **NSE** [40] and **NIST.Net** [41], the latter was chosen due to its simplicity, robustness and great number of configurable parameters. More details on network emulation and the possible configurations of NIST.Net can be found in [41].
- A traffic generator to create synthetic traffic into the emulated network. **Packet Generator**, which comes along the Linux kernel, was tested but didn't fit our stability and robustness tests. It also didn't permit the use of any statistical distribution other than a "constant" one. The **Traffic Generator 2** [42] (TG2) was used instead, and was modified to include provisions for the use of a pareto distribution, necessary for easily generating self-similar traffic. Self-similar traffic is explained in section 6.2.4 and more details on the use of TG2 can be found in [42] and [43].
- A good pseudo-random number generator for use with stochastic processes that needed it (namely, for the packet generation). This work used an implementation of the **Mersenne-Twister** uniform pseudo-random number generator [44]. Given that there are no truly random number generators presently, this work chose a generator with a period of $2^{19937}-1$ samples, only after which the samples become predictable and lose randomness.
- A monitoring tool to gather all the data and organize it. **RRDTool** [45] was chosen due to its ability to use fixed-size databases (also called round robin databases - RRD) and its ease of configuration, update and graphic generation (for monitoring).
- A robust statistical tool for analyzing all the collected data. The **R** [46] tool was used to analyze the data and to build some of the graphics shown in this work.

A testbed was also needed to run all this. The **Next Generation Networks Testbed** (NGN Testbed) from the QoSWARE project [47] was kindly conceded for these experiments. The infrastructure is explained in more detail in the following section.

6.2.1. Testing Infrastructure

All Measurements have taken place in the laboratory of the ***Grupo de Pesquisa em Redes e Telecomunicações*** (GPRT – Networks and Telecommunications Research Group), at the Federal University of Pernambuco (UFPE). This work used the Next Generation Networks Testbed (shown in Figure 6-1), available for network tests and measurements of the QoSWARE project in GPRT. The testbed is composed of 8 computers and two Fast Ethernet switches. All machines had the following hardware configuration:

- Athlon 1.333 Ghz Processor
- 256 MB Random Access Memory
- 20 GB Hard Disk
- 4 Network Interfaces

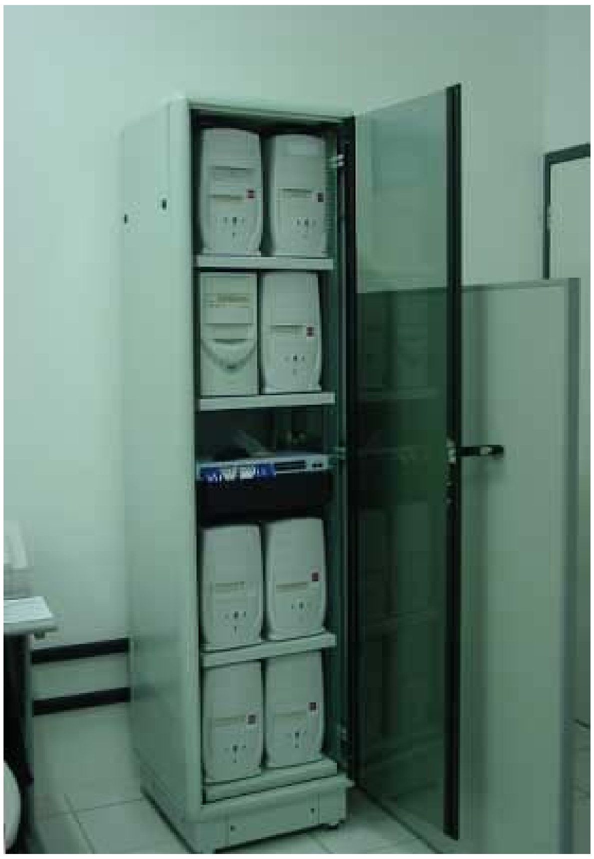

Figure 6-1: Next Generation Networks Testbed in GPRT

The machines have two network connections with each switch, which are connected with each other. This allows for the configuration of many different network topologies. Although the connection between the two switches is in high speed (multiple interfaces with load sharing), special care has been taken not to allow a connection between two network interfaces (of the computers) passing through both switches, to avoid the extra delay.

The Mersenne-Twister random number generator was installed on all machines of the testbed, instead of the standard random number generator function available in Linux. It was installed prior to the compilation of the programs. No machine on this test needed to generate more than 4 billion random numbers for all the tests, which is several orders of magnitude smaller then the period of the generator used.

6.2.2. Network Topology

This work needed a network topology with certain characteristics to properly analyze the proposed alterations for clock synchronization infrastructure. Two NTP machines (a client and a server) were needed. The server had to be synchronized to international standards and could be connected to an atomic physical clock.

A simple Diffserv network should separate the client and the server, with two routers used to connect them. The routers should be connected to each other through a single link with a 40ms delay. NTP traffic between client and server should share network resources with other Internet flows (from many other networks connected to each router), which would bring common Internet conditions (packet loss and jitter). Grouping each NTP machine with the networks connected to their respective router forms a network topology known as dumbbell, as shown in Figure 6-2. Here, the applications mark their packets with the correct DSCP, without the need of an Edge Router to remark packets or police/shape the traffic. No bandwidth brokers are studied as well.

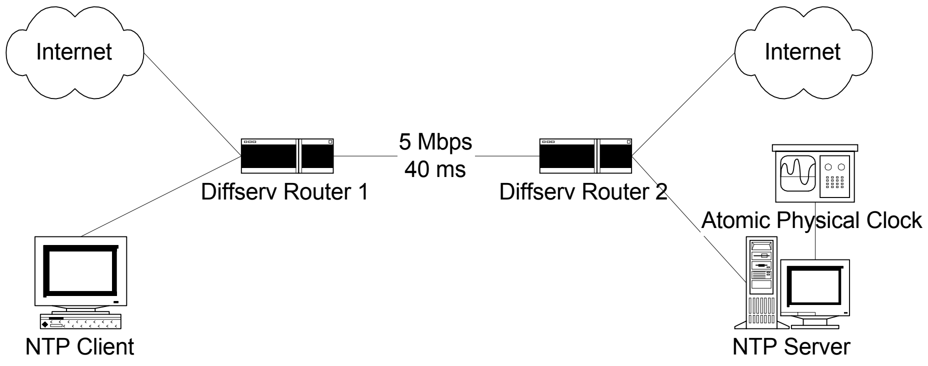

Figure 6-2: Studied Network Topology

This was the intended network topology and needed a few odd things: regular users exchanging traffic (thus making the tests uncontrolled and irreproducible), a link that would delay packets by 40ms connecting the routers and an atomic physical clock (which costs a few thousand dollars) or a GPS physical clock (which costs around US$ 700 but requires carefully planned building installation facilities) to make the NTP server furnish quality time. No radio time transmitters were available for usage in Brazil, therefore radio clocks could not be used.

This case study used a different network topology to emulate the intended topology. On the implemented topology, a few items were added or used to replace others (see Figure 6-3). In replacement to the external networks, each router had one machine connected to it to act as a traffic generator and as a traffic receiver. A dedicated link between NTP client and server was added to accurately measure the offset between their clocks (this wouldn't be possible on a regular network with geographically distant machines). A network emulation machine was used in replacement to the link between the routers. The emulator delayed all packets passing in any direction by exactly 40ms, with no artificial jitter introduction. The observed jitter resulted only from queuing (as verified with and without the traffic generators running).

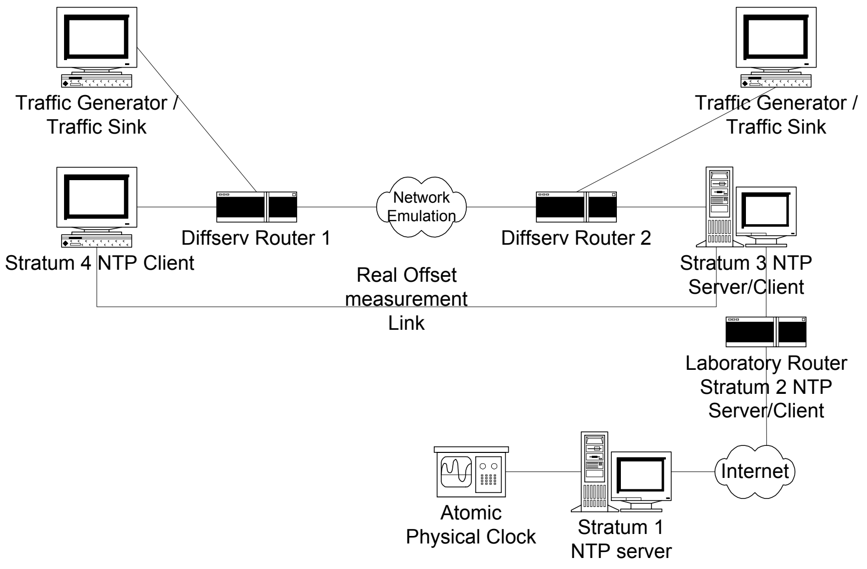
Traffic Generator / Traffic Sink
Traffic Generator / Traffic Sink
Network Emulation
Diffserv Router 1
Diffserv Router 2
Stratum 4 NTP Client
Stratum 3 NTP Server/Client
Real Offset measurement Link
Laboratory Router Stratum 2 NTP Server/Client
Internet
Atomic Physical Clock
Stratum 1 NTP server

Figure 6-3: Implemented Network Topology

Server synchronization was also different. Without our own physical clock, an external time reference was used, through the Internet. Since our network was protected (using network addresses for local IP networks), we needed a machine to function as a bridge for NTP (between the local network and the Internet). The laboratory router was used for that. It used a remote stratum 1 NTP server and the server of the experiment (now stratum 3) used the router to synchronize its clock. In order to make sure the stratum 3 server furnished a good, trustable time, the stratum 2 and 3 NTP servers were configured with a short polling interval ($P = 32\,\text{seconds}$) to the stratum 1 and 2 servers, respectively, and begun running several days before the start of the experiment (in order to let local clock corrections stabilize). The marking of the DS Field (as detailed in Chapter 4) by the application required a code recompilation of NTP between sets of test with different configurations of the experiment.

6.2.3. Advanced Routing and Traffic Control

Iproute2 is a set of tools for providing advanced routing capabilities in Linux. They are user-level tools that interact directly with kernel-level modules. It can be used for incoming and outgoing traffic policing, multipath routing, load balancing, tunneling, **Virtual Private Networks** (VPNs), multicasting, **queuing disciplines**, bandwidth management (e.g., traffic shaping) or router redundancy (with more machines). Some of these features also require other complementary tools, but none were needed in our experiment.

Queuing disciplines are forwarding schemes that provide means (an abstraction and a mechanism) to organize outgoing traffic on an interface. They can be divided in two main groups: **classless** queuing disciplines and **classful** ones (with classes that correspond to queues or other queuing disciplines). Since queues can be used alone or without queuing disciplines, queuing disciplines can in turn use other queuing disciplines and the difference between queues and queuing disciplines is very subtle, this work will not differentiate between them. Here are some commonly available queues and queuing disciplines in Linux:

- **Byte First-In First-Out** and **Packet First-In First Out** (BFIFO/PFIFO) – This is a simple queue (generally referred to simply as FIFO or **drop-tail**) where all traffic is treated equally. The size of a BFIFO is set in bytes and the size of a PFIFO is set in packets.
- **Random Early Detection** (RED) – it is a queue that will drop packets before the queue is full. RED can be viewed as a physical queue with three virtual queues. It uses two thresholds, and calculates an estimation of the queue size to decide upon the dropping or queuing of an incoming packet. This estimation of the queue size is calculated using a weighted average of the last estimate and the last real size of the queue, with a very small weight for the last real size of the queue. If the estimated queue size goes above the first threshold, each packet has a small probability of being discarded. If it goes above the second threshold, a big percentage of packets will be discarded. This will decrease the average delay of packets and will drop packets from random flows a little before congestion occurs, thereby preventing high congestion levels and preventing global synchronization of TCP congestion control (which is very bad for throughput and causes delay and jitter).
- **Generic Random Early Detection** (GRED) – it is a modified version of RED that can be configured with any number of virtual queues (up to 16), any

drop probability for each virtual queue and a different weight for calculating the average queue size.

- **Weighted Round Robin** (WRR) – it is a queuing discipline that distributes bandwidth among classes. Each class will have an associated bandwidth and WRR will distribute the total bandwidth in a round robin scheme. This provides fairness, but may increase average delay of traffic.
- **Priority First-In First-Out** (Priority FIFO) – it has three queues, with growing preemptive priority. Each queue starves the lower priority queues until it is empty. The classification of packet into queues uses the TOS Byte from the IP header and has a pre-defined configuration that cannot be changed. This queuing discipline will not be use alone in a DiffServ network.
- **Token Bucket Filter** (TBF) – it is a simple queuing discipline that provides a mechanism to limit traffic according to a rate, also allowing limited bursts that exceed the rate. Is consists of a bucket of tokens, where tokens are inserted at a rate that specifies the allowed traffic rate and each forwarded packet consumes tokens from the bucket (or are discarded if there are not enough tokens for the packet). The tokens can fill the bucket up to a limit, which allows momentary bursts limited by the bucket size. The bucket size and token rate are configurable and TBF is used for traffic shaping.
- **Stochastic Fair Queuing** (SFQ) – it is a queuing discipline that uses a great number of FIFO queues. Each traffic flow will be sent in a specific FIFO queue, but by using a hash to make this mapping to the number of queues is not specified by the number of flows and many flows can be treated by the same queue. Traffic is sent like round robin, giving each queue its turn to transmit. This should prohibit queues with very high traffic to drown other queues, thus providing fairness among them. To prevent unfairness among flows that share a queue, the hash is changed frequently. SFQ is only recommended for overloaded links and brings no QoS, only fairness. Since it keeps state information from flows (for the hashing), it doesn't scale well and should be used on customer networks only.
- **PRIO** or **PQ** (Priority Queuing) – it is a simple queuing discipline that subdivides traffic into subclasses and any treatment can be configured for each subclass. It can be seen as a PFIFO queuing discipline that can be thoroughly

configured. Each subclass has a priority level. The method of classification on each subclass can use any field (or combination of fields) from the IP header.

- **Class Based Queuing** (CBQ) – it is a classful queuing discipline that can be thoroughly configured. It uses a truly hierarchical set of subclasses, and each subclass may correspond to a flow or a flow aggregate, based on any field (or combination of fields in the IP header). CBQ can also be used for traffic shaping. Each subclass needs a configured rate, which can be isolated (it won't lend traffic to other classes if it is idle) or shared (it will lend if its traffic is not being used) and bounded (it cannot borrow bandwidth from other classes even if they are not fully used) or unbounded (it can borrow bandwidth from other classes if they allow it). Parameters set for a class will be shared by its subclasses. Each subclass has a priority level and may have a maximum allowed burst. Internally, CBQ functions as a round robin process looking for a packet to be sent that goes through all the classes in a decreasing priority order and checking if each class has a packet to transmit and if its rate configurations are being respected. After one packet is selected and transmitted, the search begins again from the higher priority class. The shaping is done with idle time calculations for each class. On the packet classification, CBQ always tries to match first the inner classes from a hierarchical set of subclasses.
- **Hierarchical Token Bucket** (HTB) – it is a classful queuing discipline that also uses hierarchical queues. It was based on CBQ, but uses token buckets to shape traffic instead of idle time calculations. It cannot make some of the guarantees of CBQ, but is simpler to use.
- **Ingress** – it is a classful queuing discipline for incoming traffic on an interface. It can be used to police incoming traffic (including traffic destined to the local machine) before it reaches the IP stack. It uses token buckets to shape traffic and classes cannot borrow traffic from one another.

There's also a mechanism to be used with a traffic classifier that can help using DiffServ. **DSMARK** is a mechanism that allows an Edge router to mark or re-mark incoming packets with a specific DSCP. This experiment did not use any Edge Router, and therefore DSMARK was not needed.

For more details on the use of advanced routing in Linux, see [38]. For other references on DiffServ for Linux, see [37].

6.2.4. Self-Similar Traffic Generation

A traffic generator was used to deliver synthetic traffic in the network to share the main link with NTP traffic (i.e., server as background traffic). Many stochastic processes were studied to describe different types on Internet traffic, but it was proved that it could be best described as having a **fractal** (or **self-similar**) nature [48]. Even when observed in very different timescales (e.g., from 1ms interval samples to 100s interval samples [48]), Internet traffic graphics look nearly the same. The fact that it looks different on extreme timescales (e.g., interval samples below 1ms), which is called **multifractal** behavior, was not considered relevant to this work.

A metric was established to measure the level of self-similarity on network traffic, called the **Hurst** parameter (H). This parameter ranges from 0.5 (no self-similarity) to 1.0 (complete self-similarity). It is important to notice that the more fractal the traffic is, the burstier it is, which is bad (greater queuing, greater average delay, smaller average throughput) for efficient network utilization. Some proposals have been made to reduce the self-similarity of Internet traffic, but significant changes could only be achieved by changing TCP behavior significantly (as the proposal in [49]).

It has been shown [50] that any level of self-similarity can be achieved through the aggregation of on-off sources (2-state Markovian sources, where **on** means transmitting a burst at a constant rate and **off** means not transmitting) using a pareto distribution to regulate the on or the off durations.

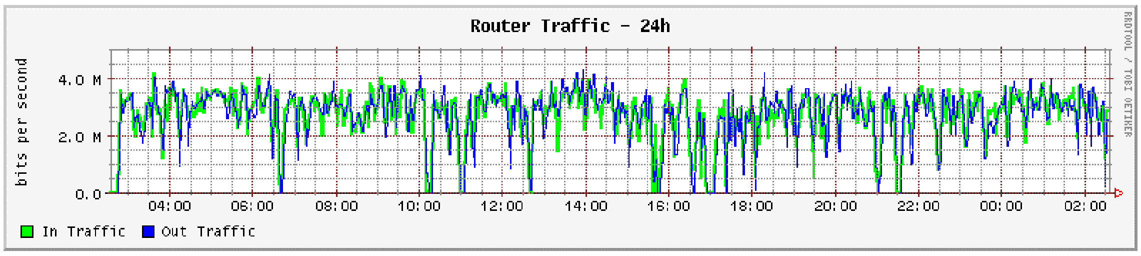

Figure 6-4: Typical day of traffic generation

The traffic generator used, TG2, was changed to allow the use of pareto distributions. Our synthetic traffic generation was done to make use of the emulated link in both ways. This work used the aggregated generation of 20 UDP sources (on each traffic generator, with packets always destined to the other generator) for a maximum possible rate of 7.2Mbps (in each way, in case all sources from one side transmit at the same time) and an average generation of 3.6Mbps. It was configured to result in a significant, but common (i.e., normally observed in internet traffic studies) level of self-similarity, measured as a Hurst parameter of $H = 8.5$. In the EF scenario, one of the sources transmits EF traffic (in each generator). A typical period of 24h of the generated traffic is presented in Figure 6-4.

6.2.5. Data Gathering

Data gathering was done using the RRDTool [45]. Each hour was considered one replication of the experiment. Data was sampled every 60 seconds and its absolute value was averaged over an hour. The main variables monitored were the estimated offset between the NTP client and the stratum 3 server (estimated by NTP itself, which runs using the emulated network link) and the real offset (measured with a specific tool in the NTP package using a direct dedicated link) between the NTP client and its server.

The stratum of the NTP client (to monitor synchronization losses), and the traffic passing through the interfaces of the routers (to monitor traffic generation activity) were also monitored for checking the integrity of the experiment.

All the data was analyzed using the R statistical tool [46]. It was also used for plotting the graphics needed by the analysis. Some graphics didn't need any analysis and were plotted using RRDTool.

6.3. Results

Though we gathered the real and the estimated offset, the latter is only relevant on the light of the real offset. Therefore, this work analyzed a new variable, the offset estimation error, calculated as the difference between the estimated and the real offsets.

A preliminary test was run to decide how many replications were necessary to achieve statistical guarantees on the tests [51]. The decision was based on the maximum error acceptable for each test, calculated as a percentage of the average. The confidence level used for this analysis was 90%. Table 6-1 compares the results for each analyzed variable in each scenario.

Table 6-1: Determining Number of Replications

Analyzed Variable	Type of Traffic (scenario)	Sampled Average	Maximum Error	Number of Replications
Real Offset	Best Effort	0.009817	15%	117.8108
Real Offset	Expedited Forwarding	0.004889	15%	49.98287
Real Offset	Hot-Potato Forwarding	0.003376	15%	32.98568
Offset Estimation Error	Best Effort	0.002501	15%	110.1464
Offset Estimation Error	Expedited Forwarding	0.002707	15%	42.46485
Offset Estimation Error	Hot-Potato Forwarding	0.002421	15%	33.31118

As a result of the comparison, the test was made with 120 replications for each scenario, which resulted in 5 days (120 hours) running each of the three scenarios.

During the test, on the scenario that treated NTP traffic as best effort traffic, the client lost synchrony with the server 12 times during the whole experiment. As NTP continues disciplining the local clock on these cases, the local clock did not wander without control. Since this is a common event on production networks, it was considered normal operation and the samples were normally accepted for the experiment. No synchronization loss occurred on the other scenarios.

The maximum jitter was measured in each scenario for the type of traffic NTP was using with simple "ping" (with an option to change the DSField). After the run of each experiment, 1000 ping packets were sent and the difference between the maximum and minimum observed RTT (round-trip time) delay was taken and divided by two to represent the one-way delay. This difference is an estimate of the maximum possible jitter that can be observed by a traffic flow, though the jitter itself is the difference observed between two adjacent packets of a flow. The results are shown on Table 6-2.

Table 6-2: Maximum Possible Jitter Estimate

Type of Traffic	Observed Jitter
Best Effort	1.5 ms
Expedited Forwarding	0.275 ms
Hot-Potato Forwarding	0.095 ms

The jitter observed on HPF traffic is still a little higher than what it should be, but it is believed that this was due to the use of general-purpose computers (where network activities are treated by software) to act as routers. In all types of traffic, the jitter is limited to the fact that only two routers were used, and on a given network topology the number of routers and the configuration of the queues (especially BE queue, with very varying configuration) affect the observed jitter. The three scenarios did not show any significant change in the loss rate of the background traffic, and the number of total lost packet did not differ significantly (approximately 0.499% for all tests). This was due to the use of UDP traffic, which doesn't use any congestion control mechanism. The goal of the background traffic was to simply imitate general Internet behavior for NTP, and not to analyze how NTP would affect other types of traffic.

On Figure 6-5 we can see a comparison between the resulting offsets between client and server of each type of traffic, computed as absolute values. The vertical bars represent the standard deviation of the data weighted by the trust interval coefficient [51], computed based on the confidence level of 90%. Undoubtedly, NTP performs systematically better as EF or HPF traffic than it does as BE traffic. As expected, HPF traffic represents a little improvement over EF traffic.

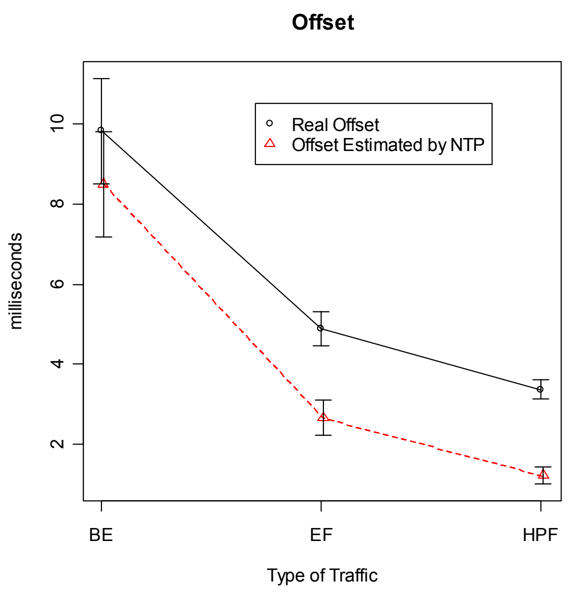

Figure 6-5: Offset Comparison

On the same figure, we see the mean estimate of the offset. As this only looks meaningful relatively to the real offset, we must show another graphic that presents the average of the difference between the real offset and its estimate. Figure 6-6 shows this difference, computed as absolute values.

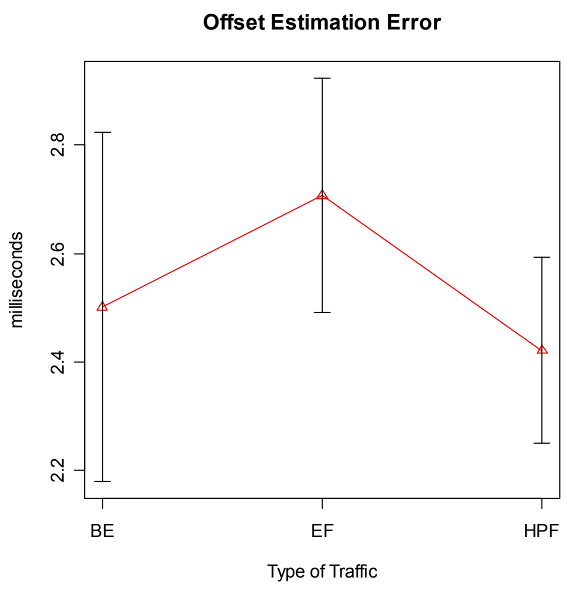

Figure 6-6: Comparison between offset estimations

Here, it is clear that the change in the type of traffic used by NTP did not significantly affect its offset estimate. Though the average has a little increased in EF traffic, its variation is smaller, resulting in a very small aggravation of this estimate. Even though the jitter caused by EF traffic is small, and therefore the variation of the estimate represented by the vertical bars on EF traffic is smaller, this jitter is more unpredictable, since some packets receive a minimum delay and some others receive a higher one. HPF traffic, however, showed a little improvement on the mean and on the variation of the estimation.

This case study clearly shows that NTP very much benefits from the use of the EF PHB and the HPF PHB. The latter showed itself reasonably better than the first.

Chapter 7 - Conclusions and Future Work

This chapter concludes this work by commenting its main benefits, detailing the contributions it made and exposing some of the future work.

7.1. Conclusions

This work shed light on some issues that may improve clock synchronization systems for networked computers and draws some interesting results. It has focused the distribution of time for synchronization of clocks in networked computers and proposed some network configurations based on the use of Quality of Service for dealing with it, including a novel treatment of packet forwarding. This proposal has been evaluated by performing an extensive case study based on network emulation and the proposal was statistically validated.

It was shown that the use of the Expedited Forwarding PHB or the proposed Hot-Potato Forwarding PHB can help ensure a permanent synchronization of a computer using NTP and that it can also improve the quality of such synchronization. There was no need to alter the behavior of NTP, with the exception of the usual markings of the packets required to introduce QoS.

Many networks can benefit from the use of this proposal, which can decrease the overall use of NTP traffic on the Internet (it is well known that in order to recover from its synchronization losses NTP raises its bandwidth) and improve its accuracy.

7.2. Contributions

The main contributions of this work can be resumed as follows:

- Alterations made on the Traffic Generator 2 for allowing the generation of self-similar traffic. The synthetic generation of fractal traffic required the use of a pareto distribution to regulate the time between traffic bursts. This distribution has not been previously implemented in the original TG2 code and this work implemented it in order to allow the performing of the case study.
- The proposal of the Hot-Potato Forwarding PHB. This forwarding scheme was proposed and explained in this work. Its implementations mechanisms and configurations were explained as well. The HPF was proposed for use with time synchronization applications and can also be used for network link characteristics monitoring.

- The proposal of a framework for dealing with time synchronization applications and improving the confidence of synchronization systems over the Internet. This framework suggests the use of the Expedited Forwarding PHB or the Hot-Potato Forwarding PHB for all time messages exchange, thus giving the synchronization application (namely, NTP) some guarantees over the jitter. This work also suggests some configurations in NTP to help it yield a better accuracy in synchronization.

7.3. Future Work

Some future work have been identified an evolution of this work. They are explained below:

- A broader study of the peak-to-mean relation for NTP traffic. It is common knowledge among system administrators that busy NTP servers only receive more than twice the average bandwidth in terms of requests when some of the clients are badly configured. While this remains true for busy servers, it has not been evaluated for less used servers and our infrastructure did not allow us to make such an evaluation. Intuitively, one would assume that the lower the number of clients, the higher the peak-to-mean ration of the traffic going to a single server. The literature shows that no effort have been done to analyze the rate on routers that treat many flows destined to many different servers. Self-similar traffic studies on the Internet show that the statistical multiplexing does not provide much gain from superposing many flows into a single one, and that traffic statistics present with many advantages and limitations, but none of these studies focused on NTP traffic in backbone routers. This future work is only relevant for the use of NTP with the EF PHB and has no relation with the use of the HPF PHB for NTP traffic.
- A study of the effect of this proposal on TCP traffic. While the objective of the use of synthetic traffic generation was to provide an Internet-like environment for how routers treat NTP traffic, the use of TCP for generating synthetic traffic is interesting for analyzing how NTP will affect other traffic flows (especially loss) when using the EF PHB or the HPF PHB. Given that NTP uses very little bandwidth, it is expected that the change in NTP traffic treatment will not change TCP behavior significantly, i.e., the number of TCP sources that will enter a "slow start" due to losses caused by the use of very small extra amounts of EF or HPF

traffic will be insignificant. It is however important to verify such a statement as TCP remains a major Internet traffic user (with more than 80% of its share).

Chapter 8 - References

[1] MILLS, David L. Adaptive Hybrid Clock Discipline Algorithm for the Network Time Protocol. **IEEE Transactions on Networking**. Volume 6, number 5. October 1998.

[2] CENTRO UNIVERSITÁTIO FIEO. **Normas para a Apresentação de Trabalhos Acadêmicos**. [ABNT/NBR-14724] Elaborated by Maria Luiza Rigo Pasquarelli. 2002.

[3] SILVA, Ivan M. **Rede de Sincronismo à Hora Legal Brasileira**. Technical Manual. http://pcdsh01.on.br (last accessed in August 2003). July 2002.

[4] MILLS, David L. **Network Time Protocol (Version 3) Specification, Implementation and Analysis**. RFC 1305. March 1992.

[5] POSTEL, J. **Daytime Protocol**. RFC 867. May 1983.

[6] POSTEL, J.; HARRENSTIEN, K. **Time Protocol**. RFC 868. May 1983.

[7] LEVINE, Judah et al. **NIST Computer Time Services: Internet Time Service (ITS), Automated Computer Time Service (ACTS) and time.gov Web Sites**. National Institute of Standards and Technology Special Publication 250-59. May 2002.

[8] X/Open Company Ltd. **DCE 1.1: Time Services**. Technical Standard. October 1994.

[9] MILLS, David L. **DCN local-network protocols**. Network Working Group. RFC 891. December 1983.

[10] MILLS, David L. **Network Time Protocol (NTP)**. RFC 958. September 1985.

[11] MILLS, David L. **Simple Network Time Protocol (SNTP) Version 4 for IPv4 IPv6 and OSI**. RFC 2030. October 1996.

[12] TANENBAUM, Andrew S. **Computer Networks**, 3rd edition. Prentice Hall. 2001.

[13] Internet Engineering Task Force. http://www.ietf.org, last accessed in August 2003.

[14] PAXSON, V. et al. **Framework for IP Performance Metrics**. RFC 2330. May 1998.

[15] ALMES, G.; KALIDINDI, S.; ZEKAUSKA, M. **A One-way Delay Metric for IPPM**. RFC 2679. September 1999.

[16] ALMES, G.; KALIDINDI, S.; ZEKAUSKA, M. **A One-way Packet Loss Metric for IPPM**. RFC 2680. September 1999.

[17] BRADEN, R. et al. **Integrated Services in the Internet Architecture: an Overview**. RFC 1633. June 1994.

[18] SHENKER, S. et al. **Specification of Guaranteed Quality of Service**. RFC 2212. September 1997.

[19] WROCLAWSKI, J. **Specification of the Controlled-Load Network Element Service**. RFC 2211. September 1997.

[20] BRADEN, R. et al. **Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification**. RFC 2205. September 1997.

[21] BLAKE, S. et al. **An Architecture for Differentiated Services**. RFC 2475. December 1998.

[22] POSTEL, J. **Internet Protocol**. RFC 791. September 1981.

[23] NICHOLS, K. et al. **Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers**. RFC 2474. December 1998.

[24] HEINANEN, J. et al. **Assured Forwarding PHB Group**. RFC 2597. June 1999.

[25] JACOBSON, V. et al. **An Expedited Forwarding PHB**. RFC 2598. June 1999.

[26] NICHOLS, K.; JACOBSON, V.; ZHANG, L. **A Two-bit Differentiated Services Architecture for the Internet**. RFC 2638. July 1999.

[27] RNP - Projeto NTP. **Lista dos servidores NTP Stratum 2 no Brasil**. http://www.rnp.br/cais/ntp/ntp_stratum2.html, last accessed in August 2003.

[28] MILLS, David L. **Experiments in Network Clock Synchronization**. RFC 957. September 1985.

[29] MILLS, David L. **Algorithms for Synchronizing Network Clocks**. RFC 956. September 1985.

[30] MILLS, David L. Internet Time Synchronization: the Network Time Protocol. **IEEE Transactions on Communications**. Volume 39, number 10. October 1991.

[31] DEERING, S.; HINDEN, R. **Internet Protocol, Version 6 (IPv6) Specification**. RFC 2460. December 1998.

[32] XIAO, X.; NI, L. M. Internet QoS: A Big Picture. **IEEE Network**. March 1999.

[33] LIAO, R.; CAMPBELL, A. T., **Dynamic Core Provisioning for Quantitative Differentiated Service**. Technical Report. Columbia University. April 2001.

[34] CASE, J. D. et al. **Simple Network Management Protocol (SNMP)**. RFC 1157. May 1990.

[35] CARSON, Mark. **Application and Protocol Testing through Network Emulation**. Internetworking Technologies Group. September 1997.

[36] NTP. **ntp.org: Home of the Network Time Protocol**. http://www.ntp.org, last accessed in August 2003.

[37] HUBERT, Bert et al. **Linux Advanced Routing and Traffic Control HOWTO**. http://lartc.org (last accessed in August 2003). December 2002.

[38] SOURCEFORGE. **Differentiated Services on Linux**. http://diffserv.sourceforge.net/, last accessed in August 2003.

[39] RIZZO, Luigi. DummyNet: a Simple Approach to The Evaluation of Network Protocols. **ACM Computer Communication Review**, volume 27, number 1. January 1997.

[40] FALL, Kevin; VARADHAN, Kannan. **The ns Manual**. The VINT Project. http://www.isi.edu/nsnam (last accessed in August 2002). August 2003.

[41] NIST. **NIST Net Home Page**. http://dns.antd.nist.gov/itg/nistnet/, last accessed in August 2003.

[42] MCKENNEY, Paul E.; LEE, Dan Y.; DENNY, Barbara A. **Traffic Generator Software Release Notes**. SRI International and USC/ISI Postel Center for Experimental Networking. January 2002.

[43] PCEN – Postel Center for Experimental Networking. **PCEN Home Page**. http://www.postel.org, last accessed in August 2003.

[44] MATSUMOTO, Makoto; NISHIMURA, Takuji. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. **ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation**. 1998.

[45] OETIKER, Tobias. **RRDTool Manual**. http://www.rrdtool.com (last accessed in August 2003), November 2002.

[46] R-PROJECT. **The R Project for Statistical Computing**. http://www.r-project.org, last accessed in August 2003.

[47] QoSWARE. **Gerenciamento de QoS no MIDDLEWARE para Aplicações em Tempo-Real**. http://www.cin.ufpe.br/~gprt/qosware, November 2001.

[48] LELAND, Will E. et al. On the Self-Similar Nature of Ethernet Traffic (extended version). **IEEE/ACM Transactions on Networking**. Volume 2, number 1. February 1994.

[49] SIKDAR, Biplab; VASTOLA, Keneth S.; KALYANAMARAN, S. **On Reducing the Degree of Self-Similarity in Network Traffic**. November 2002.

[50] WILLINGER, Walter et al. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. **IEEE/ACM Transactions on Networking**, volume 5, pp. 71-86. February 1997.

[51] JAIN, Raj. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. John Wiley & Sons. 1991.

Appendix

Abbreviations and Acronyms

ABNT	Associação Brasileira de Normas Técnicas
ABR	Available Bit Rate
ACTS	Automated Computer Time Service
AD	*anno Domini*
AF	Assured Forwarding
BB	Bandwidth Broker
BE	Best Effort
BIH	*Bureau International D'Heure* (International Time Bureau)
CBR	Constant Bit Rate
CN	Core Network
DiffServ	Differentiated Services
DSCP	Differentiated Services Codepoint
DTS	Distributed Time Service
EF	Expedited Forwarding
GMT	Greenwich Mean Time
GPS	Global Positioning System
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
LSR	Label Switched Router
MAC	Medium Access Control
NTP	Network Time Protocol
PER	Path Error Rate
PHB	Per Hop Behavior
PPM	Parts Per Million
PPS	Pulse Per Second
RSVP	Resource Reservation Protocol
QoS	Quality of Service

SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TAI	International Atomic Time
TCP	Transmission Control Protocol
UBR	Unspecified Bit Rate
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
VBR	Variable Bit Rate

Glossary

anno Domini	The Common Era. All the years since year 1.
Common Era	All the years since the year 1.
Core Network	The communication sub-network, i.e., the whole path from source to destination of a transmission, excluding the end nodes.
Differentiated Services	See Chapter 4.
Drift	The variation of skew with time (second derivative of offset with time)
Integrated Services	See section 2.5.1.
Leap second	A second inserted or deleted from UTC in the end of a month to correct it. See section 2.2.
Multiplexing	Consists on the passage of two or more connections on a channel or link, permitting a gain on bandwidth utilization if the connections have variable bit rates, since not all connection will transmit at maximum rate simultaneously.
Multiprotocol Label Switching	See section 2.5.3.
Network Time Protocol	See Chapter 3.
Offset	The difference between the times marked by two clocks.
Simple Network Management Protocol	A protocol for managing IP networks, first defined in [34].
Skew	The variation of the offset with time (first derivative).