TRABAJO ESPECIAL DE GRADO

PROPOSITION AND SIMULATION STUDY OF A NOVEL HEAVY HITTER DETECTION MECHANISM.

Prof Guía: Andrea Bianco
Tutor Industrial: Dario Rossi

Presentado ante la Ilustre
Universidad Central de Venezuela
Por el Br. Miguel Ángel Cárdenas Araujo
para optar al Título de
Ingeniero Electricista

Caracas, 2008

Dedicatory

To all the people that gave me support and friendship during my studies in my country and outside of it, to my family that has always been there for me in every moment, and finally to all the people to whom this work can be useful.

Acknowledgments

I want to give my must sincere thanks to the Professor Dario Rossi, who has been my principal guide during the development of this work, who has shared with me his ideas and advices and was the one that support me in my arrival and establishment in the city of Paris and at “The École Nationale Supérieure des Télécommunications (ENST)”.

I give my thanks to the Professor Andrea Bianco, who made possible that I could realize my project outside of the “Politecnico di Torino”, so in this way was offered to me the opportunity to work on an interesting project and to know the France's culture and language.

I give my thanks to FIDETEL for providing the financial support for my Master of Science studies, and I give my thanks to the three institutions that received me during my studies: “La Universidad Central de Venezuela”, “Politecnico di Torino” and the “ENST de Paris”.

Finally to my family, that have gave me all of their love every day, I hope I'm able to retransmit it to them in return. Thanks Mon and Dad for all your support, without it this experience outside home could not have been possible, thanks to my uncle Armando my aunts Yelitza, Yaritza and Soveida, and my brothers and cousins for been always there for me.

Table of Contents

Dedicatory....ii
Acknowledgments....ii
Acronyms....vii
Summary....ix
Introduction....1
Motivations....1
Context of this work....2
Dissertation Outline....3
General Objective....3
Specific Objectives....3
Methodology:....4

Chapter 1....6
Context Information....6
1.1 Basic Concepts....6
1.1.1 The Internet....6
1.1.2 The Best-Effort....7
1.1.3 Transport Protocols Used:....8
1.1.3.1 UDP....8
1.1.3.2 TCP....9
1.1.4 Quality of Service (QoS)....10
1.1.4.1 Integrated Services (IntServ)....11
1.1.4.2 Differentiated Services (DiffServ)....13
1.1.5 Definition for Heavy Hitter (HH)....15
1.1.6 Heavy Hitters and the QoS....15
1.2 Related Work....17
1.2.1 Sample and Hold....17
1.2.2 Mono-stage and Multistage Filters....18
1.2.3 Runs bAsed Traffic Estimator (RATE)....18
1.2.4 Accel-RATE....19
1.2.5 Hierarchical and Multidimensional Heavy Hitters....19
Chapter 2....21
Heavy Hitter Selection Mechanism....21
2.1 Mechanism Overview....22
2.2 The Counting Scheme (CS)....25
2.2.1 Cumulative Counting Scheme....26
2.2.1.1 Alternative Idea for the Cumulative-CS....28
2.2.1.2 Memory Requirement....29
2.2.1.3 Enhancement for the memory use....30
2.2.2 Mean Counting Scheme....30
2.2.2.1 Memory Requirement....33

2.2.3 Vector Counting Scheme..33
2.2.3.1 Memory Requirement..35
2.3 Low Level Details..36
2.3.1 Instant-checking or snapshot..36
2.3.2 Counting Packets..38
2.3.3 Per flow state information..39
2.3.4 The Lists..39
2.3.5 Computational Complexity..40
2.3.6 Floating Point vs Integer numbers..41
2.3.7 Over-Measuring..41
Chapter 3..42
Performance Evaluation..42
3.1 Presentation of the simulation scenario..42
3.1.1 Some reasons for the scenario configuration..46
3.1.2 Flows Identification..47
3.1.3 Detailed parameter selection..48
3.2 Flows behavior in the simulation scenario..50
3.2.1 Reading the horizontal axis of plots..52
3.3 Parameters to study in the model..53
3.4 Mechanism evaluation..54
3.4.1 The HHs-Table..54
3.4.1.1 The m-ranks and the a-ranks in the HHs-Table..55
3.4.2 How the model sees the flows..58
3.4.2.1 Groups occurrence in the HHs-Table..59
3.4.2.2 Individual occurrence of the flows in the HHs-Table:..61
3.4.2.3 The m-ranks distribution 62
3.4.2.4 The a-ranks distribution..65
3.4.2.5 M-ranks Vs a-ranks..67
3.4.3 Selected HH..69
3.4.4 Fidelity for the 1st m-rank on the Selected-HH..72
3.4.5 The effect of over measuring in the fidelity..76
3.4.6 The mean m-rank..76
3.4.7 Accuracy Vs Fidelity..79
3.5 Cumulative, Mean and Vector counting schemes..80

Conclusions..83
Bibliographic References..85
Bibliography..87
Glossary..90

Appendix A: “Guide for installation and general use for the DropTailQueueExplorable queue-type in the Omnetpp Network Simulator”.............93

Appendix B: “Resumen en español”..110

Illustration Index

Figure 2.1 Mechanism Activation/Deactivation..21
Figure 2.2 Checking Process..24
Figure 2.3 Cumulative-Counting Scheme Example..28
Figure 2.4 Mean-CS Example...32
Figure 2.5 Behavior of some flow in time...35
Figure 2.6 More realistic queue checking...37
Figure 2.7 Flow-chart of a Historical Check...38
Figure 3.1 Simulation Network...44
Figure 3.2 Per group and flow mean throughput...52
Figure 3.3 Per Group PDF of Mean Thrpt...59
Figure 3.4 PDF of Groups at HHs-Table with N=6..60
Figure 3.5 PDF of Groups at HHs-Table with N=3..60
Figure 3.6 Per Flow Occurrence..61
Figure 3.7 m-ranks PDF for N=6...64
Figure 3.8 m-ranks CDF for N=6...64
Figure 3.9 m-ranks PDF for N=2...64
Figure 3.10 m-ranks CDF for N=2...64
Figure 3.11 PDF a-ranks for N=6...66
Figure 3.12 CDF a-ranks for N=6...66
Figure 3.13 PDF a-ranks for N=2...66
Figure 3.14 CDF a-ranks for N=2...66
Figure 3.15 m-ranks PDF for the groups..68
Figure 3.16 PDF of a-ranks for groups...68
Figure 3.17 CDF of a-ranks for groups...68
Figure 3.18 PDF of m-ranks for SHH..70
Figure 3.19 Fidelity varying NHchk Vs N...73
Figure 3.20 Fidelity varying N Vs NHchk...73
Figure 3.21 3D fidelity varying N and NHchk...74
Figure 3.22 Fidelity varying NHchks Vs Dt..75
Figure 3.23 Fidelity varying N Vs Dt...75
Figure 3.24 Mean m-rank varying NHchk Vs N..78
Figure 3.25 Mean m-rank varying N Vs NHchk..78
Figure 3.26 Accuracy Vs Fidelity varying NHchk..80
Figure 3.27 Accuracy Vs Fidelity varying N..80
Figure 3.28 Accuracy Vs Fidelity for Mean-CS...81

Index of Tables

Table 3.1.1 TCP flows principal parameters....46
Table 3.1.3.1 Links Capacities....48
Table 3.1.3.2 Servers UDP and TCP Streams Bit Rate....49
Table 3.4.1.1 HHs-Table example....55
Table 3.4.6.1 Ideal mean m-ranks....77

Acronyms

ACK:	Acknowledge packet
CBR:	Constant Bit Rate
CDF:	Cumulative Distribution Function
CEO:	Chief Executive Officer
CQTI:	Check Queue Time Interval
CS:	Counting Scheme
DoS:	Denial of Service
DS/DiffServ:	Differentiated Services
ECN:	Explicit Congestion Notification
FIFO:	First In First Out
HCP:	Historical Checks Period
HH:	Heavy Hitter
HHH:	Hierarchal Heavy Hitter
IETF:	Internet Engineering Task Force
IntServ:	Integrated Services
IP:	Internet Protocol
ISO:	International Organization for Standardization
MCQL:	Maximum Crossing Queue Latency.
MPLS:	Multi Protocol Level Switching
MSS:	Maximum Segment Size
NHchks:	Number of consecutive Historical checks.

OSI: Open Systems Interconnection

PDF: Probability Density Function

PHB: Per Hop Behavior

QoS: Quality of Service

RATE: Runs bAsed Traffic Estimator

RESV: Reservation

RNG: Random Number Generator

RSVP: ReServation Protocol

Rspec: Resources specification

SHH: Selected Heavy Hitter

SLS: Service Level Specification

TC: Traffic Conditioning

TCA: Traffic Conditioning Agreement

TCT: Two-run Count Table

TO: Time Out

Tspec: Traffic specification

Miguel Angel Cárdenas Araujo

PROPOSITION AND SIMULATION STUDY OF A NOVEL HEAVY HITTER DETECTION MECHANISM

Academic Tutor: Andrea Bianco. Internship Tutor: Dario Rossi. Torino. Politecnico di Torino. III Facoltà di Ingegneria. Laurea Specialistica in Ingegneria Telematica. Internship Institution: École Nationale Supérieure des Télécommunications. 2008.

Keywords: Heavy Hitter, Traffic Engineering, Queue.

Summary: In the present dissertation is proposed a new mechanism with three (3) implementable variations (counting schemes) for the detection of the most significant so called Heavy Hitters (flows whose load inside a flow-set is very relevant) and or a group of them.

The mechanisms is to be applied in egress router queues; it consist in evidencing every certain period of time (called HCP) what is considered as the HHs applying some counting scheme. Moreover, inside each HCP is done a fix number of consecutive queue checks separated by a time interval; at every check, is count the number of packets for each of the found flows and the information of all those checks is related by the counting scheme; that information is only valid during one HCP. The counting schemes are: (1) cumulative, (2) vector, (3) mean.

The mechanism is evaluated in a controlled simulation scenario in where can be found TCP and UDP flows with a variety of bit-rates. In particular is denoted the attention to the cumulative scheme which in implementation considerations shows to the fastest and less memory consuming of the schemes.

In the study was determined that increasing the number of queue checks improves the selection of the HHs by the mechanism; on the other side, increasing the time interval between checks has not much effect in the selection. Where compared the results of the Cumulative-CS with those offered by the Mean-CS and was found that the performance in finding the most relevant flows was better for the cumulative counting scheme which is also the fastest and less memory consuming of the schemes.

Introduction.

Motivations.

The amount of data transfers in the computer networks is affected by some elements like: the evolution of personal computers, the necessity to create computer networks that can offer more services for the users that use such devices and the software evolution which each time looks forward to offer more flexibility, quality and interactivity to the users. All of those elements will have as a principal cost the increase of data information to transfer.

By another side, the evolution of communication networks like telephony and the willing to integrate more and more different devices, services and networks in the minimum variety of networks architectures as possible, has brought all together, the need to create network infrastructures with faster transmission data rates. Thus the development of new services, devices and software applications and increase of data to transfer can be summarized as a sustained increase in data rates for the different types of communications.

In networks like the IP based networks which are packet-switched, the transmission of information is done by using data-packets, so when regrouping packets that belong to one communication and that go in one sense (i.e. in a transmitter-receiver direction) is talked about *data flows*; flows that can be identified by information found in their conforming packets. Well, the mentioned flows could take different proportions of bandwidth along different sections in the network and those sections could be differently loaded with different numbers of flows. For instance the sections that belong to the core networks could hold millions of flows the last mile sections could hold some hundreds or thousands of flows each one representing a communication.

Consequently, it becomes essential for network service providers and enterprise network operators to be able to identify those flows that have anomalous behavior. Indeed such flows could represent a considerable proportion of the traffic that can potentially represent serious problems in performance for the majority of the other flows: for instance they could create bottlenecks in portions of the net, or transfer data speed decreases in the other flows; as a consequence this will result in low quality communications, or other words decrease in the overall performance of the network due to those flows commonly called Heavy Hitters (HH).

The effective detection of those HHs can lead to create mechanisms that could handle those flows like redirecting them in network portions that could be less congested or send some warnings to the sender, or simply discarding their packets or maybe activate mechanisms that could disable or extinguish the flows from the very transmission points.

The present dissertation aims to evaluate a novel mechanism for the detection of Heavy Hitters flows that avoid maintaining excesive per-flow state.

Context of this work.

This thesis work was done as an internship at the "École Nationale Supérieure des Télécommunications" (ENST) of Paris, France; in the context of TIGER, a Celtic project of the Eureka pan-European framework, for industrial and academic research. TIGER, whose acronym stands for *`Together IP, GMPL and Ethernet Revisioned'*, aims at devising an autonomic and self-managed network architecture based on the above technologies. The work carried in the current thesis constitutes one of the necessary building blocks for the development of self-managing features: indeed, efficient and distributed ways of individuating Heavy Hitter flows (i.e., the primary responsible of congestion) are needed in order to automate the reaction of the networking devices (i.e., dynamic flow re-routing, dynamic link resizing, etc.)

More specifically, this dissertation is devoted to the analysis of a novel algorithm that avoid maintaining excesive per-flow state, and that is based only on local and instantaneous information available at a node (i.e., counting the number of packets that active flows have in the queue, number of active flows in the queue, queue length, etc.) that could yield to effective selection of HH flows.

Dissertation Outline

The present section as could be seen contains the introduction to this dissertation, the general and specific objectives of the present work as well as the methodology used.

The first chapter contains the general concepts that surrounds the context of this work for a better understanding as well as the historical background in the subject.

The second chapter presents a description of the novel Heavy Hitter Detection mechanism proposed, it gives a high level overview as well low level details for it.

The third chapter contains a simulation scenario in where is evaluated the mechanism proposed in a wide variety of aspects, for then give the respective conclusions.

General Objective

Study existent techniques and/or mechanisms for heavy hitter flows detection in data Networks and propose a novel one that can detect the most relevant HH and/or a group of them, evaluate it in a simulative scenario, study and express its bounds, requirements, and benefits.

Specific Objectives

For the development of this work and accomplishment of the general objective, the principals objectives that where need to accomplish were:

- Study, use and comprehension of the Omnetpp simulator from its source code level, this with the objective of being able to modify the code when necessary in order to integrate the new code with already written code, as well as being able to see its bounds, and obviously to understand the syntax proposed in order to use it to generate the simulations.

- Study the INET Framework, which is a plug-in/extension of Omnetpp that provides those network protocols that are not natively integrated in Omnetpp and can help to reproduce more real scenarios to the simulations.
- Study of the C++ programming language, which is the base language in which the Omnetpp code is written.
- Research of documentation about throughput measuring and heavy hitters detection and related information to this work.
- Study and use of some tools normally included in Linux distributions like gnuplot, awk, grep, octave, shell scripts, etc, necessary to automate some simulations and analyze, modify, rearrange and show the data obtained in them.
- Creation of algorithms and modules to detect the most relevant heavy hitter flow or/and a group of them inside a flow set.
- Generation of a simulation scenario that can help to measure the capabilities of the written algorithms.

Methodology:

First Phase.

The first phase was constituted by the group of general studies and information recompilation for the C++ language, the Omnetpp simulator and its INET framework, and the research of background information on heavy hitters detections.

Second Phase.

Initial design of the modules and algorithms for the detection of the most relevant heavy hitter, creation of testing programs to ensure that those algorithms are working properly as well as the respective debug of all of them; use and learning of the Linux tools already mentioned in the specific objectives, to manage the data obtained.

Third Phase

Corresponds to creation of the scenario in which to taste the HH detection mechanism.

Fourth Phase.

Corresponds to forward modifications of the HH detection code and generation of new ideas for it, based on the results of the previous phases, this feedback between the last three phases is done to detect unexpected problems and in order to tonify the detection HHs machine.

Fifth Phase.

Results analysis and presentation of them in the present written report.

1 Chapter 1

Context Information

1.1 Basic Concepts

1.1.1 The Internet.

Normally is said that the term *Internet* refers to all the networks that cooperate to form a seamless network for their collective users using for it the Internet Protocol *IP* (which will be briefly explain later) this include federal, regional, campus and a big variety of different networks. Moreover as soon as some non IP-networks providers realized its potential, they wanted to offer their services to their clientele, so they developed methods for connecting with this IP network (for example: using some *gateways* for it, or even using other protocols that are able to transport IP). Thus in practice the compose of all this networks is what is actually "commonly" more referred as the Internet. [10]

For what concerns the business model, it has not a owner, a president or a Chief Executive Officer (CEO), in other words there is not a single authority figure for the Internet as a whole, but their constituent networks may have their CEOs.

Similarly no one pays for the Internet, there's not an Internet corporation that collects fees from all the Internet networks users or providers. Instead every one pays for their part, this is, networks are created and decide how to connect themselves. For instance a school pays for its Internet connection to some regional network which in turns pays to a national provider for its access.

For what concerns more closely the technical aspects, the Internert is the largest existing packet switched network, where a data packet refers to a self contained independent entity of data composed basically by two logical entities, the header or Protocol Control Information (PCI) and the *payload.* The first entity contains the information that can make that the packets can be transmitted in a packet switched network (is thus relevant for the network apparatuses such as routers) and the second entity refers to the real message to be transmitted (thus relevant for users and/or their applications). [18]

The IP protocol refers to a connection-less (this is, it does not requires the setup of a virtual connection between a source and a destination for transporting the information) and unreliable (it is not ensured that the information sent will arrive to its destination or that it will be not duplicated) data-oriented protocol.

The IP protocol is used for packets routing in packet-switched networks; in other words packets or datagrams are used to transmit the information which are routed independently. As a completing comment, datagram is the most commonly used term for packets when taking about this protocol.

The IP protocol does not corrects errors in datagrams, but it can discard those packets in which errors are found using a checksum field in its header, it can support fragmentation of packets, usually useful when passing through networks in which the maximum transfer unit size of the packets are different.

1.1.2 The Best-Effort

Is a classification of low priority network traffic, used especially in the Internet [17]; it consists in that some some user/application starts sending a data flow over a network (without needing a permission before transmitting), and this flow has to do its best effort to arrive to the destiny.

Since in different portions of the network the bandwidth is shared differently by different flows, the original flow needs to take its own proportion of bandwidth at each portion of the network; the objective is that the flow arrives to the destiny without being a priority for it some delay or loose of sequence or lost of datagrams. In simple words the objective is that the flow arrives to the final destiny at any price and that it can do it as soon as possible.

Best effort services can exist at various layers in the ISO/OSI model. For example, whereas IP is a network layer protocol, UDP is a transport layer protocol and Ethernet and HDLC (high level data link control) are data link layer protocols.

In the case of the IP Best Effort is in effect a paradigm that has been tremendously successful in supporting data applications till now days, but there are many that think that the present Internet architecture does not provides and/or will not provide enough support for the so-called real-time applications in future. For example applications for audio and video [2], so there exist some architectures that can perform reservation of resources like IntServ and DiffServ (sections 1.1.4.1 and 1.1.4.2).

1.1.3 Transport Protocols Used:

The simulations realized in the present work use as network layer the IP protocol and as transport layers the TCP and UDP protocols.

1.1.3.1 UDP

The *User Datagram Protocol*, defined in RFC768; refers to a simple connection-less and unreliable protocol that provides multiplexing/de-multiplexing to the service application layer.

Some of the principal characteristics of the UDP connections is that it has not a native-control mechanism for reducing or increasing its packets sending frequency in the case of packets loosing, it does not offers packet retransmission; it is message-based, this is, it moves complete messages from the application to the network layer and its transmission speed is normally the highest as possible (best effort).

Typical applications that run over UDP are streaming multimedia (like music and videos) and Internet telephony.

1.1.3.2 TCP

The *Transmission Control Protocol*, initially defined in RFC793; refers to a reliable and connection oriented transport layer protocol, that unlike UDP retransmits the packets that have been lost in the network. In it, the connections are established by the three way handshake protocol, it is stream based, which means that the data is sent to the application with no particular structure [19].

At the source host, TCP buffers the incoming bytes for the sending application, they are sent to the network layer when the size of the buffer has reached the MSS (maximum segment size) or after the expiration of a time out (TO).

The TCP checks that no bytes are corrupted by using a checksum, if an error is verified is considered as packet loss, it implements two mechanisms to control the rate of the transmitted streams: *`flow control'* and *`congestion control'*, the first one limits the sending rate in order to prevent overflow at the receiver's buffer, and the second prevents network congestion by reducing the sending rate after indication of packet loss.

The first release or version of TCP was named *Tahoe* and implemented a very simple congestion control algorithm, that will be briefly explained [6]: the congestion window size `*cwnd*' denotes the maximum number of unacknowledged segments that the sender can tolerate before transmitting a new segment. During the initial phase called `*slow start*', its value is increased by one for each acknowledgment (ACK) received. This results in exponential growth of the congestion window. When `*cwnd*' exceeds a threshold `*ssthresh*', the server enters into the `*congestion avoidence*' phase, during which `*cwnd*' is increased by one for each window of `*cwnd'* segments acknowledged. Each TCP segment sent to the receiver triggers the start of a TO, which is canceled upon reception of the positive ACK for this segment. The TO expiration is used to indicate that a segment has been lost, in that case, the server reenters the `*slow start'* phase with `*cwnd*' equal to 1 and `ssthresh' equal to its half. Because of the linear increase of the congestion window size during congestion avoidance and its sharp decrease upon the expiration of a TO, TCP congestion algorithm is often called an `Additive Increase Multiplicative Decrease' algorithm. Must be said that there exist some been other versions of TCP like Reno, NewReno, Vegas, Hybla, BIC, etc. DiffServ-[6]-[14]

TCP Reno [14] which is normally the most widely used now days, implements the `*fast retransmit*’ that allows immediate retransmission of thought loss segment when three duplicate ACKs are received before the respective TO expires and the `*fast recovery*’ which avoids to enter in the Slow Start phase when just a segment has been lost, so if three consecutive ACKs of the same segment are received the `*ssthresh*’ is the half of the minimum between `*cwnd*’ and `*rwnd*’ and the *cwnd = ssthresh +3*, is retransmitted the lost segment and starts a grows at the slow start style (cwnd=cwnd+1 for each ACK received in sequence) until arrives the ACK that confirms that the old lost packet was received, then *cwnd = ssthresh* and starts a grow at the congestion avoidance style (cwnd=cwnd+1/cwnd for each ACK received in sequence).

TCP supports many of the Internet's most popular application protocols and resulting applications, including the World Wide Web, e-mail (SMTP), File Transfer Protocol and Secure Shell.

1.1.4 Quality of Service (*QoS)*.

To overcome the Best Effort limitations, a number of architectures have been proposed in the attempt to guarantee a Quality of Service (QoS).

QoS is in fact an overused term with various meanings and perspectives, it has been defined in the RFC2386 [16] as a set of service requirements to be met by the network while transporting a flow from its source to its destination. In the case of packet-switched networks the QoS is often accepted as a measure of the service quality that the network offers to the application/users in terms of delay, jitter, available bandwidth and packet loss [3], some times those requirements are achievable with resource reservation. In some cases can be used the term QoS as the ability to reserve resources, but other times can be associated with over-provisioning, so then the term can take that meaning also. In cases like telephony it comprises all the aspects of a connection, such as time to provide service, echo, voice quality, loss, reliability, so different applications have different QoS requirements that can be normally expressed as parameters.

Many services models and mechanisms have been proposed to meet the demand for QoS, some of them are IntServ/RSVP, DiffServ, traffic engineering, MPLS; must be said that the best effort model offers really poor QoS, it does not guarantees any Quality of Service, there are not guarantees in delay, jitters, out of sequence packets, packets loss. Best effort can only produce some unstable QoS if are used over-provisioning mechanisms in light loaded networks.

As a remark, is important to say that there is not absolute QoS possible to be introduced by looking only at the networking side, because as a general rule all the connections are subject to physics. For instance an energy brake-out can disable network connections, damaged network devices can disable portion of networks or introduce delays to network flows, natural catastrophes that could affect in any way the network. In other words there is not an absolute mechanism that can ensure 100% QoS for connections since all is subject to physics.

1.1.4.1 Integrated Services (IntServ)

Is an architecture proposed in 1994 at the RFC1633 [15], that looks forward to add QoS to the Internet, the idea is to offer to all the services hat run over it QoS aspects similar to those found on the old telephony. Thus can be assigned if necessary a delay constraint, bandwidth, and assured connectivity to existent calls, having now as analogy that those calls are now flow streams that do not necessarily are only for telephonic call purposes.

Its principal characteristic (and its principal constraint), is that every single flow is treated independently (is flow-related); the flows state information is spread in the network, not only at the end systems but also the complaint routers must be able to reserve resources to ensure the connectivity. Thus the complexity of the network components and overhead form signaling grows.

The admission control mechanism must accepted for it is the Reservation Protocol (RSVP) [1], it can be briefly described with an example; imagine that a client wants to download a file from a server:

- The receiver `client' communicates with the sender `server' (in a way not specified by the standard) that wants a file.
- The sender sends a PATH message that will arrive to the receiver; this message contains the traffic specification `*Tspec*' (i.e, constant bit rate `*CBR*') and creates a path for the connection.
- If the receiver accepts it responds with a reservation (RESV) message which contains the resources specification `*Rspec'* to be reserved (i.e. bit rate of 4kbps and max delay of 0.01ms); this message will pass by the same path of the PATH message but in backward direction, and the nodes will chose to accept or deny the request depending of their reservation resources capabilities and if all the Rspec can be satisfied (i.e., delay).
- If accepted by all the nodes the file transmission from the sender can start, otherwise the node that does not accept the RESV message sends an error message to the receiver.

IntServ offers two service classes: *Guaranteed Service* and *Controlled load*, the first one corresponds to the acceptance of a flow if its traffic requirements can be completely assured during the connectivity of the flow (i.e. delay, minimum bandwidth); for the second one there is not delay guarantee and the resources reservation needed can be adapted to a minimum.

There are some drawbacks about this architecture, one as already said is that the complexity of the routers grows (due to RSVP) as well as the information to be handled by them which increases linearly with the number of flows having as a direct consequence that the scalability is affected. Another drawback is that the majority of flows will be for short connections like for web browsing, which will imply a big overhead caused by signaling therefore the network performance perceived by the applications will be deteriorated, especially if latency for the flows is concerned. [13]

1.1.4.2 Differentiated Services (DiffServ)

Is an architecture that as IntServ looks forward to introduce QoS to the internet, its principal difference with the last one its that it does not keep state information for each of the flows, but it can keep it for aggregates of flows.

Those aggregates of flows are grouped by classes of traffic, and the QoS parameters can be different for each of the aggregates, thus different quality can be assured for the them but not for single flows. Consequently DiffServ is not as granular as IntServ, but it does not need to reserve resources for each flow and it does not necessary needs an admission control mechanism.

This architectures reuses the Type Of Service (ToS) byte of the IPv.4 packet header and renames it as Differentiated Services Field (DS Field) which is divided in DS Code Point (DSCP) of 4bits and the Explicit Congestion Notification (ECN) of 2 bits. The DSCP offers 64 combinations and each one of them can represent a different class of traffic, from them there are 8 combinations that are compatible with the old ToS of Ipv.4, they will represent the Class Selector Code Point.

For the different classes of traffic can be designated different Per Hop Behaviors (PHBs), that refer to different treatments along the network that can be done to the packets belonging to a PHB class; normally can be configured a maximum of 64 PHBs using the different markings of DSCP, but could be used more if is enabled multi-field classification on the packets. The must used PHBs are the predefined: (1) The Expedited Forwarding (EF), (2) The Assured Forwarding (AF), (3) The Class Selector and (4) Default PHB. The (1) EF PHB has the characteristics of low delay, low loss and low jitter, these characteristics are suitable for voice, video and other real-time services or for inter-industry communication. The AF (2) allows the operator to provide assurance of delivery as long as the traffic does not exceed some subscribed rate, after it its traffic has a bigger probability of having packets drops if congestion occurs. The class selector (3) refers to the service of the old ToS of IP as already mentioned, and Default PHB (4) is for best effort traffic.

This architecture can separate providers and groups of network devices using the concept of domains; a DS domain is formed by contiguous set of DS nodes implementing a common set of PHB already set up by the provider. The marking of the packets is done at the edges of the domain, been the edges those routers that are compliant with this architecture and can communicate with different DS domains and not DS complaint domains.

When a costumer wants to belongs to a specific DS domain, he must do a Service Level Agreement (SLA) with his service provider, there will be specified the traffic class wanted or Service Level Specification (SLS), and the Traffic Conditioning Agreement (TCA) that refers to the Traffic Conditioning (TC) to which the traffic generated by him will be restricted to. SLAs can be established too between different DS peering domains, this way can be created *DS Regions*, so a network of different DS domains can be established and can be offered end to end connection services to any of the networks elements between those domains.

The TC is done by a shaper and a policer, the shaper is at the side of the generating traffic element (costumer), and the policer at the service provider side. The shaper tries to give *shape* to the traffic stream generated by the user so it can be conformed to the SLS (for instance a maximum bit rate, that could be achieved with a leaky bucket mechanism). And the policer which is at the provider side ensures that the traffic generated by the user is inside its limits, otherwise some action can be taken like remarking the packets and increasing then their drop probability. [13]

For more details about this architecture can be seen some of the next RFCs: RFC2474, RFC2475, RFC2597, RFC3246.

1.1.5 Definition for Heavy Hitter (HH)

As will be mentioned in the chapter 19, there are different types of heavy hitters. Can be said for the present context, that a HH corresponds to a individual flow which associated load (in terms of bytes or packets) inside of a flow-set is bigger than a proportion of the accumulated load of all the flow-set in a determinate period of time.

Another concepts of Heavy Hitters in the literature is:

> "A heavy hitter is an entity which accounts for at least a proportion of the total activity measured in terms of number of bits, packets, connections, etc". [5]

1.1.6 Heavy Hitters and the QoS.

Irrespectively of the QoS architecture, users will generate traffic flows, some of these will be the so called Heavy Hitters. Lets speculate on the importance of the HH detection in the different QoS architectures early described.

In the IntServ architecture, the problem of detecting the heavy hitters would be reduced as much that it could be considered a minimum problem; initially because its admission control mechanism is flow-related. Thus as an starting point, any new flow must be accepted by the nodes on the network in where it will cross with the resources it declares to consume. Then if any of those nodes considers that the resources are upon a limit the flow is simply not admitted by the admission control; as a consequence this heavy hitter will not exist.

By another side, if the flow belongs for instance to the service class *Controlled load*, is possible that in periods of time it can use more resources that what he initially declares it could consume. Then the flow can be easily detected as a heavy hitter and could be taken some action if it where necessary; since the architecture as already said is flow-related and can keep per-flow state information.

At the same time, IntServ has not been widely deployed and likely never will. So, HH maybe will still a relevant problem but in other contexts.

In the Differentiated services architecture the detecting heavy hitters problem takes a wider range to the one treated in this dissertation due to its characteristic of treating flow aggregates differently. The problem of HH detection in DiffServ is complicated by the fact that flows may have different relative priorities: as such, a HH of high-priority class should be treated differently with respect to a HH of a lower priority class.

Thus should be enabled different methodologies to treat each aggregate differently or should be used a methodology that could be expandable enough to detect the heavy hitters in the different aggregates, even in those aggregates that use a Per Hop Behavior defined by the network provider.

It must be took into consideration that a flow that has some traffic characteristics that could be considered a HH inside an aggregate cannot be necessarily a HH in another aggregate, this as an intrinsic result of this architecture in which the different aggregates have different *priorities*. Thus flows that could have similar traffic characteristics in some period of time could be treated differently by the network. Additionally the allowance of network resources for a flow of an aggregate can be different to the allowance for another flow who is in more restricted aggregate; all depends in the SLA in which the flows are subject to.

In the traditional Best Effort Internet, all the flows have the same priority to cross over the network and their behavior is controlled principally by the protocols they are subject to along the net. When taking about an end-to-end flow that crosses over different networks providers, in some of the providers, the priority factor could be included if the flow packets are encapsulated or if the flow is inserted in aggregates of flows that have less priority; but as soon they pass those transit sections they will have the same priorities as the other flows (i.e. the section of a network called the last mile).

The problem of detecting the HHs in this case (Best Effort) can be said as more homogeneous, because all the flows have the same priority. Then must be only found those flows that stand out on the others using some criteria. For instance, the ones that take more bandwidth over some threshold.

In the actuality what makes principally the best effort Internet architecture to be more used, spread, supported and developed and maybe the one that will be more used in many years to come is just that its actual grade of total utility is really predominant over the complexity necessary to make the architecture work. Thus the grade of utility versus complexity is actually not that high as in DiffServ or IntServ.

1.2 Related Work

For the process of tracking the heavy hitters can be mentioned a variety of works, some of them will be mentioned and briefly explained in this section in order to get a vision of prior works into the subject. Must be said that in order to get deeper details than the given in this section the references should be viewed.

One of the most trivial ways if not the most simple to detect the HHs is to sample all the packets that arrives to a node, and to have a per flow count on bits or packets for all the flows. Then from this count determine which flows have the bigger proportions of the node or link usage; the problem with this approach is scalability and that is not efficient in the use of computational resources.

1.2.1 Sample and Hold

Is based on packets sampling, so if a packet is sampled and the flow it belongs to has no entry in the memory, a new entry will be created; that entry is a counter held in a hash table, and it will last a time interval period. The particularity is that not all the fluxes are added, because the process of adding a new flux is done with some probability, but after a flux has a new entry all the packets that belong to that flux are counted. In summary sample and hold counts all the packets after an entry of a given flux is created. [7]

1.2.2 Mono-stage and Multistage Filters

A stage is a table of counters that are indexed by a hash function that computes a packet flow ID, so all the packets of the same flow are hashed to the same counter, in this way can be put a threshold of number of packets for a flux. If the threshold is surpassed the flux is considered as a heavy hitter, is important to say that every counter is increased every time a packed ID is being hashed in it.

But because the number of counters that can be afforded is normally smaller than the number of flows, many flows could be mapped to the same counter assigned to a larger flux and could be taken as false positives.

To reduce the number of false positives is proposed the use of parallel or serial multistage filters. In the case of parallel (which is considered by the paper's author as the best), every stage has an independent hash function and every packet flow ID is mapped to all the parallel stages, at this point, only the fluxes that surpasses all of the stages thresholds are considered as heavy hitters.

For example in a dual stage parallel filter, if a flux W is hashed in counter 4 of the first stage and 7 of the second stage and the two counters count is bigger than a threshold, this flux is considered as heavy hitter. In the contrary if a flux R is hashed in the counters 4 and 3 of the two stages respectively, and the counter 3 of the second stage does not surpasses the threshold, then this flux is not considered as heavy hitters and is ignored. [7]

1.2.3 Runs bAsed Traffic Estimator (RATE)

Is based in the idea of sampling a small portion of the arriving traffic to a node being selective and sampling more frequently those flows that have a larger proportion of the traffic, to achieve this objective are counted the two-runs. Is said that a flow has a two run, if two consecutive samples belong to that flow, the sampling not necessarily has to be for consecutive packets, they can be sampled with some probability, in this way small flow sources have low probability to get two runs.

The information of the flows that achieve those two runs is saved in a Two-run Count Table (TCT), which contains the number of two runs for each flow that has had a two run. Is used a register that will hold the last flow-ID sampled, if the next sample has a different flow-ID the register is simply reset, if the ID is the same the register will be reset too but the two run count for the flux will be inserted/updated in the TCT. For instance a four flow sequence like ADBBBBC will have two two- runs for the flow B, and a sequence as ADDDBC, will have only one two-run for the flow D, therefore the detection of two runs is a regenerative process. [9]

1.2.4 Accel-RATE

Is a work based in RATE that looks forward to improve its performance, the idea is to uniformly hash the incoming streams into X buckets or sub-streams and perform a two run count on each bucket, the principal target with this modification is to improve the estimation time used to detect the HH within respect to RATE, doing for it a trade off between memory and processing power versus estimation time, because of that this scheme needs more memory and processing power than RATE.

This speed up is basically a result that each flow corresponds to a bigger fraction of its assigned sub-stream that in the case of a unique sub-stream and therefore the probability to obtain two runs is bigger than in the case of RATE. As a characteristic of the model each sub-stream has an associated two-run register, and there is a common TCT table for all the sub-streams. [8]

1.2.5 Hierarchical and Multidimensional Heavy Hitters

In this work the authors expand the concept of a HH and say that it does not necessarily has to be just one individual flow, it can be an aggregation of multiple flows/connections that share some property, an example of property could be the hierarchy so that will be a “Hierarchical HH”. For instance IP addresses can be organized into hierarchy according to prefixes, so could be identify all a group of IP address as HH if they surpass some threshold in a node band with utilization. In this work the authors mention like another property the dimension, which will involve the combination of different protocols layers, like the combinations of IP addresses and port numbers which will represent a bi-dimensional flow. Must be said that in the present dissertation is treated the problem of bi-dimensional and mono-hierarchical heavy hitters. [5]

2 Chapter 2

Heavy Hitter Selection Mechanism

In this chapter will be described the approach proposed for the detection of the most significant so called Heavy Hitters (flows whose load inside a flow-set is very relevant) and/ or a group of them.

The mechanism is to be applied in queues (i.e. router outgoing queues), it does not , it should be triggered when in the queue (in where is to be applied) is found some activity enough to accumulate packets inside of it. If the queue is empty or closely empty, the mechanism is unable to make any valid conclusion about the HHs flows. Thus should be taken into consideration the mean queue length per periods of time, and enable the mechanism when it overpasses some “activation threshold” and disabling it when it underpasses some “deactivation threshold”.

In the figure 2.1 is exemplified the activation and deactivation of the model in time; in particular the horizontal axis represents the time and the vertical axis the mean queue length. In the figure, the growing plot exemplifies the evolution of the *mean queue length* and the horizontal plots represent the thresholds for the activation and deactivation of the mechanism. There is not given any values in the figure because is only a representation.

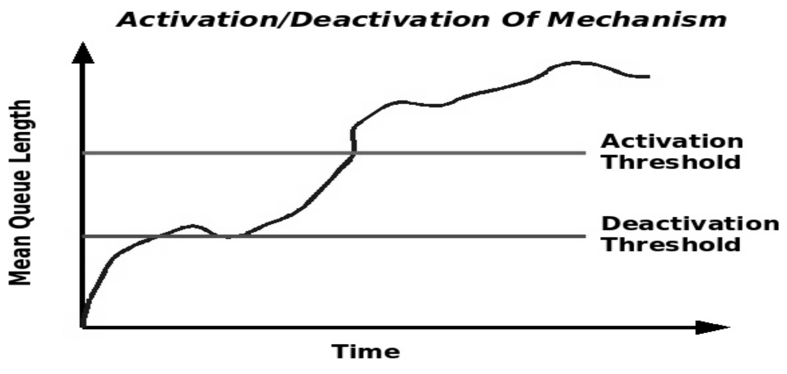

Figure 2.1 Mechanism Activation/Deactivation

In the section 2.1, is initially found a high level over-view of the methodology, in the section 2.2 are explained the counting schemes proposed and in the section 2.3 are introduced some of the low level details for the mechanism.

2.1 Mechanism Overview

Is recalled, that the aim of the present work, is to be able to identify the flows that generates the most of the traffic. To do so, the implemented method measures some instantaneous properties of the queue. These properties are observed (sampled) a number of times, then the HH selection decision is made; In this section is over viewed the general selection process and is deferred its evaluation to the chapter 3.

The general idea is to determine the number of packets and/or bytes of every flow that can be found “inside” of the queue at some instant of time. Then in another time instant is repeated the same process; this means, to determine again the number of packets of every flow found in the queue for later accumulating somehow that taken information with the information taken in the precedent interval (accumulation done using the counting scheme, section 2.2). Consequently is given relevance to those flows that according to the counting scheme are the most relevants. Is like taking “snapshots” of the flows that exist in the queue every certain time interval and color the flows that persist and are more powerful according to the counting scheme during some number of consecutive intervals.

When doing a check or “snapshot” of the flows in the queue, what is really done is to save in some memory the flow information that will identify each different flow and that allows to count how many packets this flow has had in the queue at that instant. The flow-ID could depend in the implementation, as an instance could be a hash number that hashes the identifying information of the flow as the usual form of the 5th-tuple (IP-source, IP-destination, source-Port, destination-Port, IP-protocol-type).

As a note and general concept, *hashing* represents the action of mapping some object (string or concatenation of strings and numbers) by using a *hash function* into another string/number normally smaller in length, usually with the intention of being able to identify some characteristic of the original object. The hash function should be injective to avoid that two different flows have the same hashing value.

Lets call *Check Queue Time Interval* (CQTI) to the period of time between two consecutive “snapshots” and call *Maximum Crossing Queue Latency* (MCQL) to the time taken for a packet of maximum segment size (MSS) to cross a completely loaded queue from the moment of its arrival till the moment it leaves the queue. Furthermore, the queue is of such a type, that it serves packets with a First In First Out (FIFO) scheme and all the packets inside of it are of MSS length.

Has to be said that when *saving the flow information to the memory*, is being referred to the action of adding to the memory a flow record (or simply record) that contains the flow-ID and some scalars or vector (depending on the counting scheme) that accumulates the information of the given flow. That “memory” can be seen as a Indexed list that will be called “*Check-List*” where the key is given by the flow-ID.

Two checking times will be considered as *consecutive* if in the time between them is not erased the memory that keeps the information collected from the flows. Each of the checks that are consecutive will be also called *historical consecutive checks*, because the accumulation of them takes in consideration a historical behavior of the queue.

The *number of consecutive Historical checks* (NHchks) will be argued in the simulations to be done at chapter 3, but as a basic rule to have a minimum of historical accumulated behavior at least two consecutive checks should be applied.

After a given number of consecutive historical checks all the information of the flow-IDs already saved including the information saved by the counting scheme will be erased. Then will start a new *group of consecutive checks* that will add to the memory each detected flow-ID and associate its respective counting scheme information that will accumulated at each of the consecutive checks. A new flow-ID with its respective associated counting scheme information is added to the memory of flows if at least in one of the consecutive checks this flow has had a packet in the queue.

Every period of time in which is erased the memory of the historical checks will be called *historical checks period* (HCP). This period of time is an entire multiple of the CQTI and separates as already said two groups of consecutive checks.

In the figure 2.2 is exemplified the checking process, in order to have a general and easier vision of the model. In fact there are two sub-figures, (a) and (b).

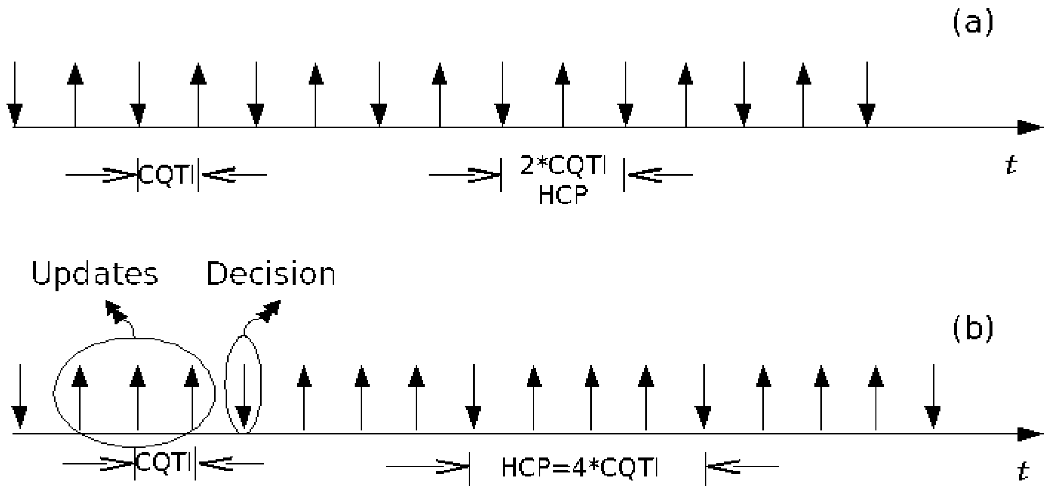

Figure 2.2 Checking Process

Lets take talk about the sub-figure (b), in its horizontal axis is represented the time and the small arrows that are perpendicular to that axis represents the historical checking instants in time. Note that the direction of the checking instant arrows is not the same, the majority of the arrows point away of the time axis and the others point in the backward direction, they will be refer as *up* and *down-arrows* respectively.

Each of the small arrows represent a check of the queue and accumulation of the historical information. The down-arrow has one main difference respect to the up-arrow and is that after the respective check is done is erased the memory associated to the flow-IDs and their historical information. Therefore, when using the concept of *consecutive checks* a sequence of down-arrow:up-arrow are not from the same group of consecutive checks, but the sequences up-arrow:down-arrow and up-arrow:up-arrow are from the same consecutive check group.

The time between any of the consecutive small arrows is the CQTI (check queue time interval), and the time between two consecutive down-arrows is the HCP (historical checks period), period which in this sub-figure (b) is four times longer than the CQTI time and in the sub-figure (a) is two times longer than the CQTI.

In practice, after every group of consecutive checks or in other words after every HCP, will be shown or highlighted those flows that according to counting scheme where the most relevants --the decision--, having as idea that those flows should be the heavy hitters; this is in fact one of the principal objectives that will be studied in the simulations, that is to know which is the proportion of cases in which those highlighted flows are really the HHs that are searched.

2.2 The Counting Scheme (CS)

The information to be taken from the flows and the way of taking that information could be different and can give different information and lead to different results. As an instance could be enough to count the number of packets of each of the flows at the historical checking queue instants, that will indicate the persistence of packets of the flows.

By another side counting the bytes of packets instead of just the number of packets could lead to selection of flows who are really more incident in the flow throughput and consequently in the proportion of traffic they take when passing through a queue. Furthermore, if all the packets have the same size as will be in our studding case for simplicity, the results of both ways will be the same, and will be easier and faster to implement just to count the number of packets.

The counting schemes proposed are:

- Cumulative-CS
- Mean-CS
- Cumulative Vector-CS

In all the counting schemes will be used two list: one is the “Check-List” which is indexed and will be used in each historical check, and the other list is the “HHs-List” which is of length N (fix natural number) is of type sorted and will be used in the last historical check of a HCP. In the lists are saved records containing the information of the flows with a format specified in the CS.

The general idea is that in the Check-List is accumulated the information at each historical check and in the HHs-List is saved the information of the most relevant flows (according to the CS) inside the Check-List at the last historical check of a HCP.

Is recommended to first refer to the Cumulative-CS section which develops the easiest CS and introduces concepts that will be retaken in the other CS sections but with less details.

2.2.1 Cumulative Counting Scheme

The idea in this CS is just to keep a logical structure or data-record that will be composed by two values, one is the flow-ID and the other is an scalar. Each record will be associated to only one detected flow during the consecutive historical checks and will be kept an order of relevance for those records.

The mentioned scalar will accumulate the number of packets that have been detected in the queue in different consecutive checking times and at the same time keep the value that gives the order or relevance for them. Relevance or importance given by sorting them with the simple idea of greater number goes first, this means for instance if a flow-ID has accumulated in two consecutive historical checks a number of 20 packets it will have more importance that some other flow-ID that have accumulated 10 packets in the same group of consecutive checks.

This records are regrouped in a indexed list (Check-List), being the key for each record: the flow-ID and which will be used in all of the historical checks. The idea with the indexed list, is to be able to found those flow-IDs rapidly enough to update the information of their accumulated packets in the queue, and of course also at each consecutive historical check to the queue.

At the end of each HCP the most relevant flows (according to the scheme) are saved in a second and smaller sorted list of length N (HHs-List), the idea is that this list should contain the first N flows that have the highest throughput in the HCP and that can be considered as HHs. According to the Cumulative-CS (as already said), the most relevant flows will be those that in cumulative have had more packets in all the historical checks.

The criteria to insert the most relevant flows in the HHs-List is to do as maximum N scans to the Check-List and found the flows in it, that have had accumulated more packets in all the checks of the HCP. A first scan consist in looking in the Check-List to the flow-record with highest cumulated packets, then a next scan will look at the next flow-record with more accumulated packets (the next flow in relevance), and like this successively till the N scans are done in the case that there are at least N flows in the Check-List.

As a general comment, remember that at the end of the HCP –this is, after the scans are done--, the Check List must be emptied for the first historical check of the next HCP, and the information of the HHs-List must be erased before the end of the next HCP in where will be re-utilized.

In the figure 2.3 below, is exemplified the idea of this CS in a way in which the same queue is checked in two consecutive historical checks (NHchk=2). For the time $t=t_{i-1}$ the queue is checked by its first time, there is got the information of the flows and how many packets for each of them. For the time $t=t_i$ is got the information of the queue at that instant and is accumulated with that of the last check, and as the cumulative-CS determines their values are accumulated. Then N scans are done to the Check-List, and are saved in the HHs-List the more N relevant flows of the HCP according to the present. CS.

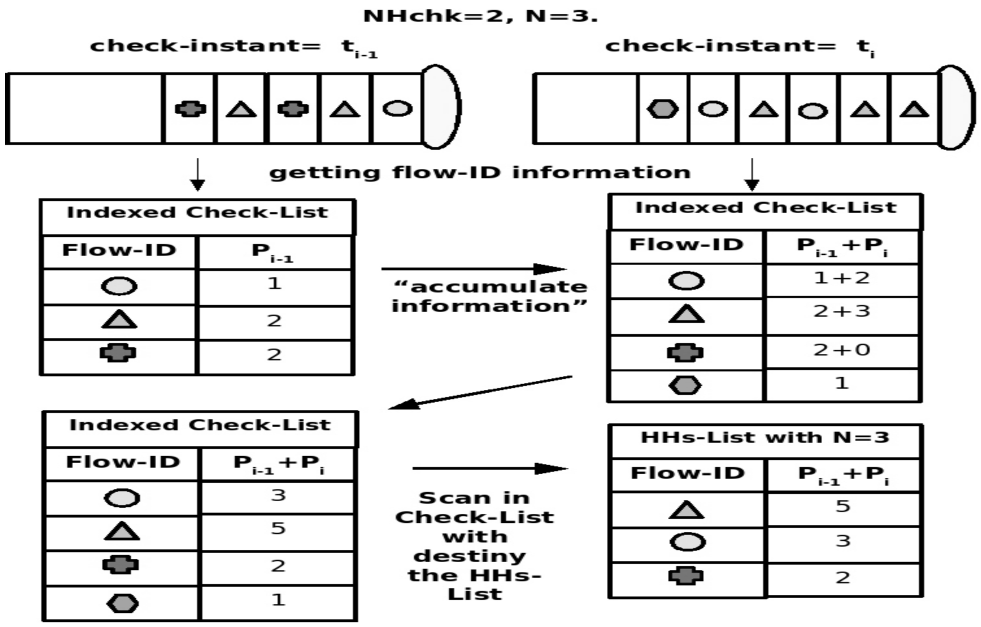

Figure 2.3 Cumulative-Counting Scheme Example

2.2.1.1 Alternative Idea for the Cumulative-CS

A different solution that could be applied, but it should be studied because it has not been test yet with this counting scheme is to apply the concept of two-runs [see chapter 1.2.3].

The idea is that instead of counting the number of packets for each of the flows must be counted the number of two-runs. Where having a two-run would refer in this case to sample with some "order or sequence" the packets inside the queue and that two consecutive sampled packets will belong to the same flow-ID.

Moreover, has to be said that the expression *two consecutive samples* does not necessarily means that the two packets are really consecutive inside the queue, it means that in the order in which the packets where sampled two of those consecutive samples taken in that specific order have the same flow-ID; the order of sampling could be aleatory but it must not take as a sample a same packet two times and must select all the packets of the queue at a checking instant.

2.2.1.2 Memory Requirement

Holding to the Cumulative-CS proposed in the section 2.2.1, will be shown the memory than could be necessary at the implementation time.

Lets call **S** to the memory associated to any of the scalar registers that will be associated to the flows; **FID** to the memory associated to any of the registers that will keep temporally the flow-ID; **MQL** to the maximum number of packets the queue can hold at some instant of time, **NHchks** to the number of historical consecutive checks done before the memory is erased and **N** to the maximum length of the HHs-List.

The maximum amount of memory that can be required in the worst case, which is when the queue is in every consecutive check at its maximum capacity and every flow has just one packet in the queue is shown in the formula 1.

$$MaxMemory=(FID+S)\times MQL\times NHchks+N\times(FID+S) \qquad (1)$$

The formula 2 indicates the maximum number of different flows that could be handled by the current model.

$$MaxNumFlow=MQL\times NHchks \qquad (2)$$

The number of flows expressed in the formula 2 does not indicates the maximum number of flows that can pass through the queue; because in the time interval between two consecutive checks could pass some packets that belong to others flows who where not detected in the checking instants.

Besides, the not detection of flows normally happens if the time between those two checks (CQTI) is considerably bigger than the maximum crossing queue latency (MCQL); so to avoid that some considerable information could be lost some trade off must be done with the time between two consecutive historical checks, this issue will be argued in the chapter 3.

Moreover, the characteristic of not detecting every flow that passes through the queue, is what makes possible that the mechanism does not keeps state information of every flow.

A forward note about the formula 2 is that if the mechanism checks that number of different flows the results given by it must not be considered; because it will mean that every checked flow has only one packet accumulated in the Check-List, so there is not possible way to select correctly the most relevant flow.

2.2.1.3 Enhancement for the memory use

Following this scheme some reductions for the memory to use could be done, like decreasing the total memory used in the Check-List to a portion of it with “k<1”.

In normal circumstances the queue should not be at its top of capacity for long periods of time, and if that case happens just overwrite in those memory locations where are found those flow-IDs whose associated counters have small values; for instance those flow-IDs that only have one packet will be substituted with those that have more. Consequently must be introduced a flag “f” that for instance is activated in those flow-IDs whose number of accumulated packets are smaller than a given number X. Then the formula 1 is rewritten in the formula 3.

$$Memory=(FID+S+f)\times NHchks\times k \ + \ N\times(FID+S+f); \quad where: \ k<1 \tag{3}$$

2.2.2 Mean Counting Scheme

The Mean-CS uses two lists, the Check-List and the HHs-List; as in the cumulative-CS the information gathered belongs only to one HCP.

On this CS the lists records are composed by three fields: (1) flow-ID, (2) an integer scalar for checked packets called **CP** (checked packets) and (3) a floating point scalar for the historical mean called “**μ**” (mean). Note: remember that the Check-List is used at each historical check and the HHs-List is used in the last historical check at the end of each HCP.

The relevance in this CS is given to those flows who have the highest values inside the μ-scalar, thus the relevance is given to the flows who have the highest mean of counted packets in the historical check periods.

The idea is that at each historical check is accumulated the number of packets found in the queue using the CP-scalar .--Note: the CP-scalar does not accumulates the number of the packets with other historical checks as is done in the Cumulative-CS--. Then in the μ-scalar is saved the on-line mean of the packets checked during the historical checks; in other words the μ-scalar uses the information of the CP-scalar at each historical check and calculates the actual mean, for it is used the formula 4.

$$\mu_i += (ChckPkts_i + \mu_{(i-1)})/i \tag{4}$$

In that formula $\boldsymbol{\mu}_i$ represents the mean for the actual check, while $\boldsymbol{\mu}_{i-1}$ represents the mean for the last check, the expression *CheckPkts* represents the number of count packets for a given flow in the present historical check and "**i**" represents a natural number that starts from 1 and that holds the number of the checks done for that flow.

Important remark: the **μ**-scalar is not equal to the cumulative of the packets in all the historical checks divided by NHchks (total number of the historical checks) for a given flow; because the mean value is calculated from the first historical check in where the flow is found till the end of the HCP. Consequently the number of historical checks in where the mean packets are calculated for a flow can be less than the NHchks in a HCP.

The remark of the last paragraph is an important difference with the Cumulative-CS, in where the mean packets for a given flow is not necessarily equal to the cumulative of the packets during all the historical checks divided by NHchks, otherwise the results of the two CSs will be the same. That difference is the motivation of this CS, in order to see if there is an important advantage of considering more or less historical checks for each flow.

After the last historical check the Check-List is scanned at maximum N times in order to detect at maximum the first N flow-records with the highest value in the **μ-**scalar. Then those flow-records are saved in the HHs-List and they are supposed to be the most relevant flows for this CS during the last HCP.

In the figure 2.4, is exemplified the idea of this CS in a way in which the same queue is checked in two consecutive historical checks (NHchk=2) inside a HCP. For the time $t=t_{i-1}$ the queue is checked by the first time, there is got the information of the flows, the number of packets for each flow is saved in the CP-scalars and the mean starts to be accumulated in the **μ**-scalars. For the time $t=t_i$ is got the information of the queue at that instant, the number of packets for each flow in this check is saved in the CP-scalars and in the μ-scalars is accumulated the mean of the packets: (a) for the flows that are already in the list the mean has information from the last check; (b) for the flow with hexagon flow-ID (which was found by first time in this check) the mean is only for the actual check and does not take into count that there was a check before. Then N scans are done to the Check-List, and are saved in the HHs-List the more N relevant flows of the HCP according to the present CS .

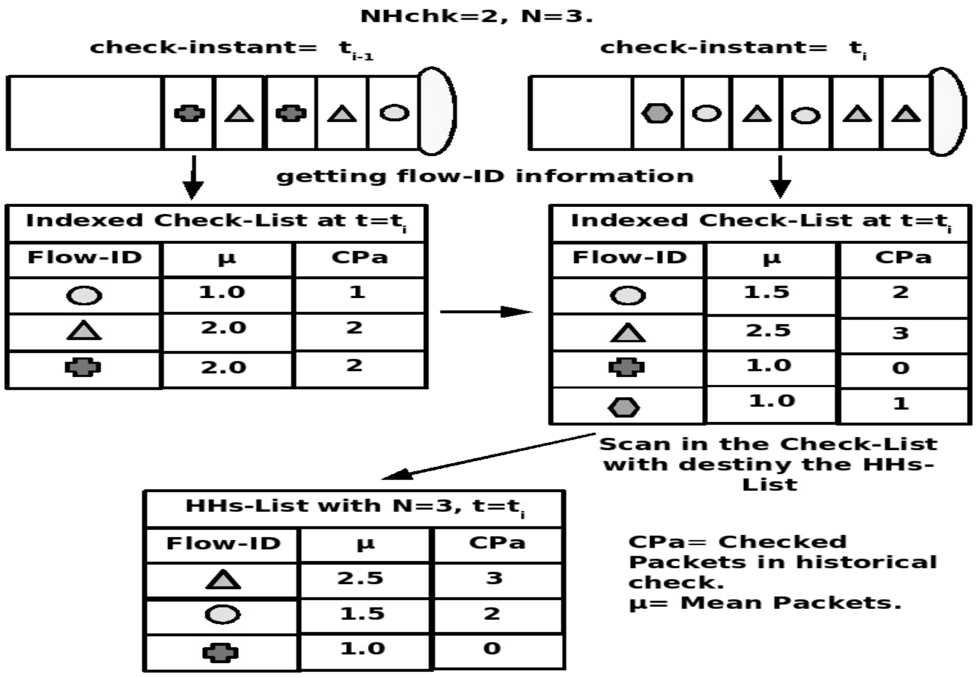
NHchk=2, N=3.
check-instant= t_{i-1}
check-instant= t_i
getting flow-ID information
Indexed Check-List at $t=t_i$
Flow-ID
μ
CPa
1.0
1
2.0
2
2.0
2
Indexed Check-List at $t=t_i$
Flow-ID
μ
CPa
1.5
2
2.5
3
1.0
0
1.0
1
Scan in the Check-List with destiny the HHs-List
HHs-List with N=3, $t=t_i$
Flow-ID
μ
CPa
2.5
3
1.5
2
1.0
0
CPa= Checked Packets in historical check.
μ= Mean Packets.

Figure 2.4 Mean-CS Example

2.2.2.1 Memory Requirement

In this scheme the memory requirement has many analogies with the requirement for the Cumulative-CS, the essential difference is that were introduced two scalars instead of one for each of the records and one of such scalars is of type float point number.

Lets call **M**, the memory associated to the scalar that holds he mean values for each of the records and **S** to the scalars that hold the counted packets also in each of the records. Using additionally the same terms as in chapter 2.2.1.2; the memory requirement in the worst case which is when the queue is completely loaded is:

$$Memory = (FID + S + M) \times NHchks \quad + \quad N \times (FID + S + M) \tag{5}$$

Similar enhancements to the memory can be done like the proposed in the cumulative-CS (see section 2.2.1.3) and the memory necessary could be rewritten like in the formula 6.

$$Memory = (FID + S + M + f) \times NHchks \times k \quad + \quad N \times (FID + S + M + f) \qquad where: \; k < 1 \tag{6}$$

2.2.3 Vector Counting Scheme

In this CS the idea is to keep an special record that contains the flow-ID and an N-dimensional vector, this vector could be for instance bi-dimensional, by so each element of the vector is composed by two variables, one variable indicates the checking instant and the other indicates the number of packets for a flow found inside the queue for the respective checking instant.

The maximum length of those vectors is determined by the NHchks, so there must be a limit in consecutive queue historical checks, otherwise becomes difficult to handle such vector. Each of the vectors plus the flow-ID represents a record so the Check-List contains now a vector for each flow representation and the HHs-List can contain for instance only the Flow-ID of the most relevant flows.

The relevance can be given to the flows in different ways; for instance give relevance to the flows that have in one particular historical check the highest number of packets, or that in mean have more packets, or that in cumulative have it. The advantage of this CS is that is kept all the information of the flows evolution in time during the HCP.

The real good point of this counting scheme is that actually it can “draw” the behavior of the flows, so can be known how is changing in time the incidence of a specific flow in the queue. Moreover, a result of keeping an historical track of the consecutive checks like in this scheme can help to detected anomalous behavior at any flow if that were the objective.

One of the anomalous behaviors this counting model could help to figure out in a easier and faster way than the Cumulative-CS [section 2.2.1], is for instance the detection of *misbehaving flows,* which refers to those flows that exceed their stipulation bandwidth limits [4]. The detection of such a type of flows can be possible thanks to that ability of this CS in keeping a history track of the flows behavior. Obviously can only be talked about the existence of this kind of flows in those network architectures that offer some QoS and establish some range of bandwidth or data rate to the flows.

For the preset work, is used the Best Effort Internet, which does not establish such ranges for the flows, as a consequence the characteristic (detection of such flows) offered by this counting model does not represent an outstanding advantage respect to the others, what it could show is the burstiness of the flows in different time instants.

This CS could be very good for keeping information when monitoring the flows, in order to give some detailed information, but that is not the objective in this dissertation. The model that is wanted to be used looks to be implemented in some fast element of a network as a router, and not doing any detail monitoring, so the objective is to reduce the memory, the processing power and time to detect the HHs as much as possible. In this manner keeping a vectorial counting for flows packets and adding the detection of anomalous behaviors becomes out of the scope, that's why this scheme will not be forwardly studied in this dissertation.

The figure 2.5, shows an example of how the info collected from a bi-dimensional counting vector could be used to describe the behavior of a flow during some time interval.

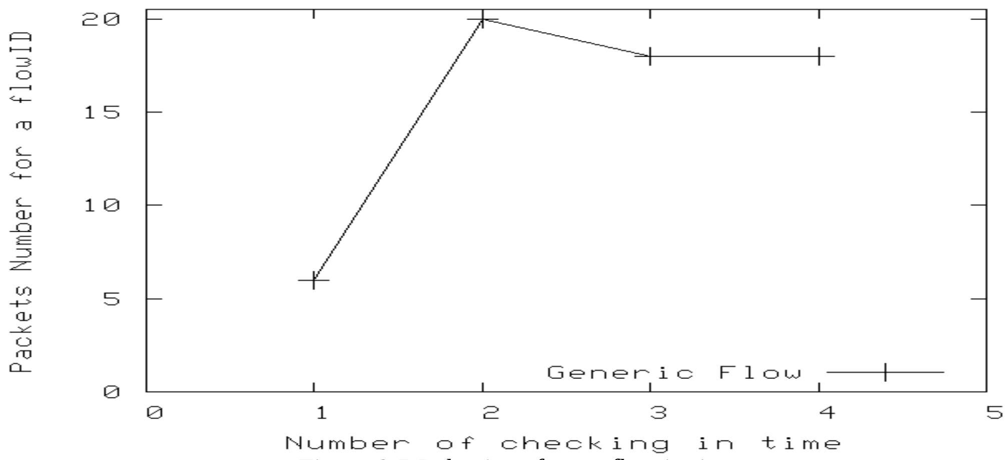

Figure 2.5 Behavior of some flow in time

In detail for the mentioned figure, the vertical axis shows the number of packets counted for a generic flow in some checking instant given by the horizontal axis; what this says is that the information kept in the bi-dimensional vector can re-generate the behavior of the flow in time.

2.2.3.1 Memory Requirement

In this scheme the memory requirement has some analogies with the requirement for the other schemes, the essential difference is that N dimensional vectors are kept for every single flow. Consequently a problem will be the scalability, because handling the vectorial information for each flow becomes very difficult in term of resources, speed and processing power if the number of flows grows.

Lets call V the dimension of the vector, lets suppose that each of the N elements of the dimension use the same memory and lets be the NHchks be the maximum length of the vector, using the same terms as in chapter 2.2.1.2; the memory requirement in the worst case which is when the queue is completely loaded is represented in the formula 7.

$$Memory = (V \times NHchks \times P + FID) \times NHchks \times MQL \qquad (7)$$

2.3 Low Level Details

In the last sections were given some low level details like the memory use in the counting schemes. In this section, will be given further details about how the mechanism works, some recommendations for the implementation and some the reasons the selection of the lists.

2.3.1 Instant-checking or snapshot

In this work the idea is to analyze by simulations the mechanism being proposed, not actually getting into truly deep details of implementation, so the instant checking can be actually done in the presented simulations.

Nevertheless, in real life the instant-checking cannot be as an idealistic as a “snapshot” in which the time is “frozen” and the checking of the packets is done. Moreover, the erase of the memory associated to flow-IDs after the HCP could only be done after the queue is completely checked and flows are treated as the counting scheme asks; so the implementation of the scheme depends on hardware capabilities. For the instant-checking will be recommended the use of *network processors* that could keep an on-line processing of the arriving packets.

With the network processor the idea could be: identify the flow-IDs of the packets during/before they are served out of the queue, so packets will still arrive to the queue and packets are served out of the queue when doing the checking. Consequently all the processing of the counting scheme can be done after all the packets are checked, then can be shown which are the flows that are supposed to be the HHs. As an example of the ideal checking, can be viewed the figure 2.6 in where is represented a queue with packets inside in different situations; so is compared an ideal checking of the queue with a more realistic one.

In the situations for the figure 2.6, the queue has packets with some flow-IDs, and is shown that in an ideal checking of the queue all the packets are simply checked in that instant; but in the more real situation, is possible that even while packets are being checked new packets arrive to the queue and that the checked packets are being served. Consequently, there must be a last packet to check in the realistic queue checking, to avoid to check packets that do not belong to the checking instant.

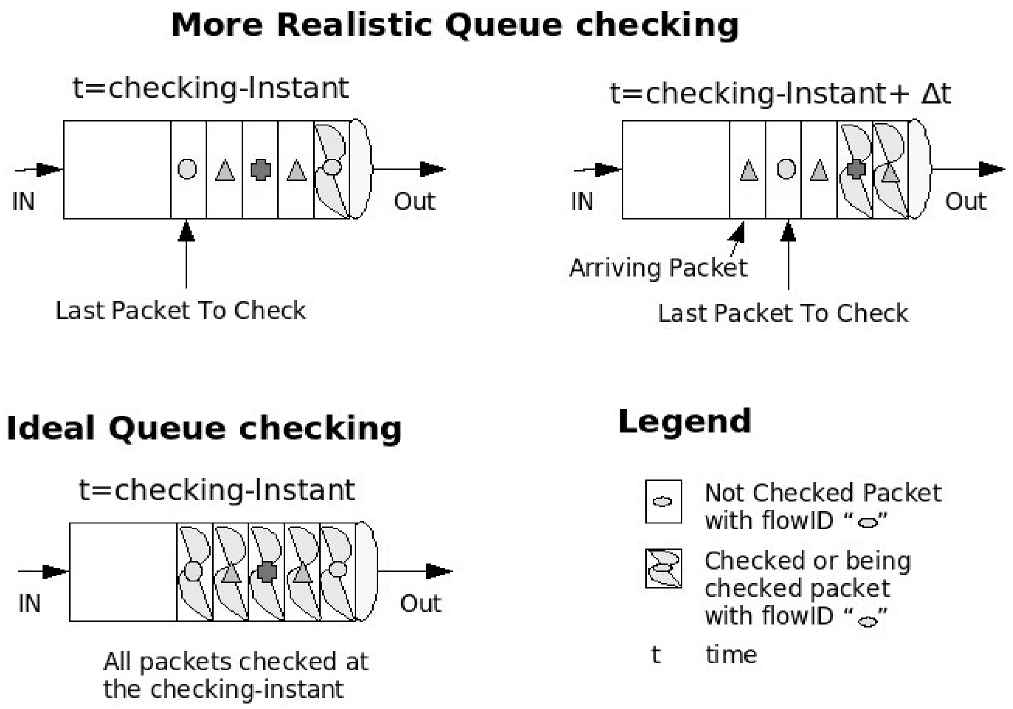

Figure 2.6 More realistic queue checking

Another idea that could be *applied* for the checking and that can be actually fast and simple, but not very "smart" in the way of using memory: is to copy all the packets headers in a memory that acts like a parallel queue and then check all the packets. That idea is more like the "snapshot", but maybe is not necessarily very smart because it will represent a waste on memory resources. Thus for this case will be needed a trade-off basically in performance and with the cost of the memory suitable for this process.

2.3.2 Counting Packets

The *counting packets* part of any counting scheme during a historical check or snap shot can be showed in the flowchart of figure 2.3. In that flow chart is shown that the flow-IDs are inserted into the Check-List, list that will be scanned later in order to insert in the HHs-List the flows that had more relevance according to the respective CS. The figure also indicates that every time a packet is found it is actualized the record that represents a flow in the Check-List, and if the flow is not in the list then is added a record for it, then (not showed in flowchart) when there are not more packets to check, the scanning of the Check-List must be done.

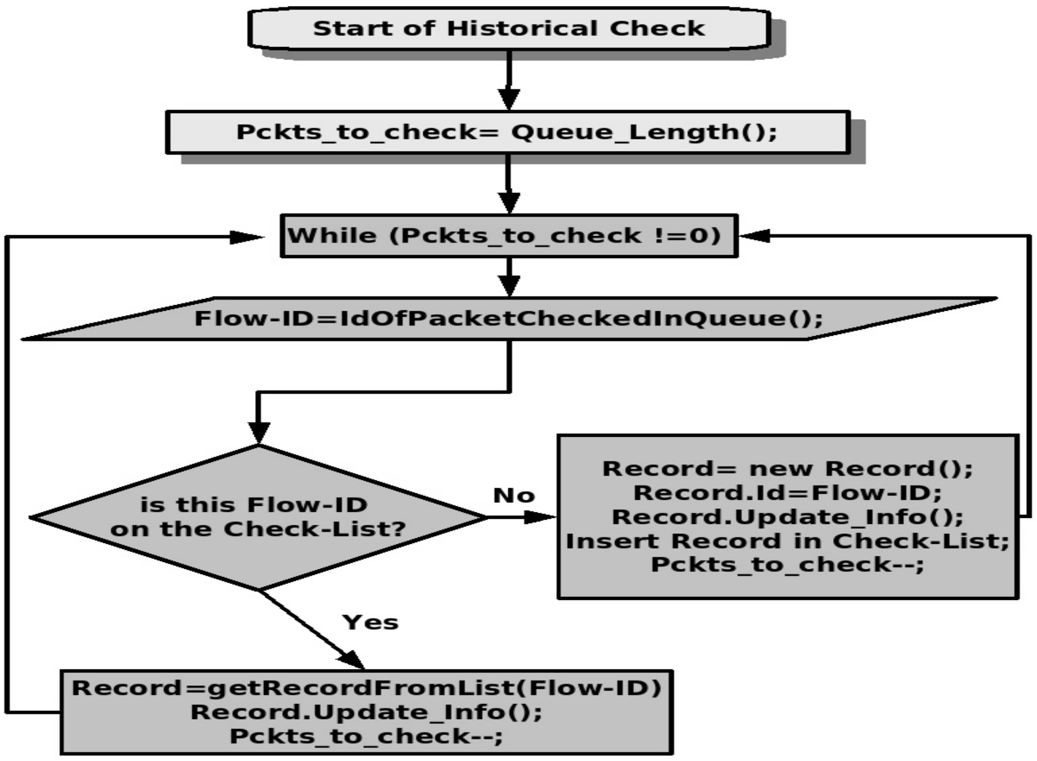
Start of Historical Check
Pckts_to_check= Queue_Length();
While (Pckts_to_check !=0)
Flow-ID=IdOfPacketCheckedInQueue();
is this Flow-ID on the Check-List?
No
Record= new Record();
Record.Id=Flow-ID;
Record.Update_Info();
Insert Record in Check-List;
Pckts_to_check--;
Yes
Record=getRecordFromList(Flow-ID)
Record.Update_Info();
Pckts_to_check--;

Figure 2.7 Flow-chart of a Historical Check

In the same figure the method “IdOfPacketToCheckedInQueue()” gets the flow-ID of the being checked packet, the method “getRecordFromList(flow-ID)” gets the record in the Check-List given a flow-ID, the method“Upadate_Info()” adds and updates the information of the record according to the counting scheme.

2.3.3 Per flow state information

In order to simplify the analysis as already said was chosen to count the packets instead of the packets bytes in the counting schemes. Lets suppose that all the packets in a queue have the same size; so in a case in which a queue is nearly or completely loaded for periods of time significantly longer than the queue's CQTI, a value for CQTI as small as its MCQL will be equivalent to have a per flow and/or per packet state information. Such a type of state information is a consequence that every packet that gets into the queue will be checked each CQTI, thus will be used a big amount of memory and processing power to keep all that information which is basically what is not wanted.

As an instance that the mechanism for HHs detection proposed does not keeps all the state information even when CQTI is equal to MCQL: is that *if the queue is not fully loaded* for the most of the time just a portion of the packets that cross the queue will be checked. Only a portion because many packets will cross the queue in a interval of time smaller than the CQTI and in instant of times that have no coincidence with the moments of the queue check. Therefore it is likely probable that flows that are conformed by a couple of packets will not be detected and no memory or processing power will be used to keep that useless information. By another side as the CQTI does not necessarily has to be as small as the MCQL not every packet will be checked at the checking instant even if the queue is almost completely loaded for periods of time bigger than the CQTI. Consequently can be said that the mechanism proposed does not keeps a per flow state information of all the flows in the network, it just keeps information of the check flows.

2.3.4 The Lists

As we have already said, the HHs detection mechanism uses two lists: the Check-List which is indexed and HHs-List which is not indexed.

The Check-List should be indexed because that kind of list is the fastest when looking at specific records and not simply looking at the records randomly, consequently there can be useful in the historical checks in where at every packet check the list should be accessed. As a general comment, the indexed lists can decrease the mean access time to any specific record than any other type of list due to the fact that its records are ordered by some key and not by their content, so is not necessary to look at the content of every record.

The HHs list has been chosen as not indexed but instead as sorted by its contents, being the sorting element the scalar that gives relevance to a given flow according to the CS (i.e. in the Mean-CS the sorting element is the μ-scalar). The idea is that the most relevant flow-records (according to the CS) are kept at the top of that list so then they can be easily accessed.

2.3.5 Computational Complexity.

When is done the last check of a HCP to a queue, the Check-List is scanned at maximum N times in order to find the first N most relevant flows on that period according to the CS. In fact was chosen to “scan” and not to “sort” that list in order to save time, memory and processing power.

In the case of sorting the records of the Check-List will be needed an operational complexity of **$O(F^2)$** where F is the length of the list, that complexity is reached for instance with the “bubble-sort” algorithm, it could be used other algorithms like the “quick-start” which has on average an operational complexity of **$O(F)\log F$**, but in the worst-case it can reach a quadratic one: **$O(F^2)$**. [11]

By another side if the list is simply scanned to find the most relevant flow, the operational complexity will be simply **$O(F)$** which is linear. If by instance the list is scanned two additional times in order to find the next two most relevant flows the complexity added will be $O(F-1)$ and $O(F-2)$, being the total complexity equal to **$O(F)+ O(F-1)+ O(F-2)$** which will be approximately **$3O(F)$** and is still linear, thus it depends in the number of elements and can be rewritten as **$O(F)$**. Consequently for N scans the complexity is linear and equal to **$O(F)$**.

As a result was selected to do scans for N elements of the Check-List instead of sorting it because the complexity is linear in the first case while in the second is quadratic. Remember that more complexity means more time and processing power is needed in order to obtain the wanted results.

2.3.6 Floating Point vs Integer numbers

In the the Cumulative-CS is used an integer scalar to count the packets and to give the relevance to the flows, meanwhile in the Mean-CS exists additionally a floating point scalar that gives relevance to the flows.

Using integer numbers is usually recommended against using floating point numbers in terms of speed; in fact when using floating point numbers and doing for instance a division of two floating point numbers, the amount of operations and time needed is more than for example an addition of two integer numbers. Consequently considerations must be taken when implementing the Mean-CS, like advantages respect to the Cumulative-CS, those details will be studied in the section 3.

2.3.7 Over-Measuring

Is important to mention that the CQTI of a queue cannot be very small, it should have a minimum value that is at least equal to its MCQL in order to avoid over-measuring. For instance if some queue is loaded at its maximum capacity and the CQTI is smaller than the time it takes to the queue to serve the last packet inside of it then that last packet could be taken into consideration two or more times depending in the CQTI. Thus that over-measuring can lead to unexpected results as a consequence that the packet will be reconsidered more times that what it should, heading maybe to exalt a flow that does not really has so many packets in the queue checks.

3 Chapter 3

Performance Evaluation.

In this chapter will be evaluated the performance of the HHs detection mechanism using for it a simulation scenario. To well understand this chapter must be understood how is the HHs selection mechanism [chapter 21]. As a general overview in the sections from 3.1 to 3.3 are described the characteristics of the simulation scenario while in the section 3.4 is evaluated the HHs mechanism performance, and finally in the section 3.5, are compared the different counting schemes proposed.

3.1 Presentation of the simulation scenario

The simulation scenario to be presented looks forward to test the behavior of the mechanism of HHs detection shown in this dissertation in a situation where can be found a group of flows of type TCP/IP, UDP/IP. The purpose is that in the scenario can be seen and studied as many characteristics as possible for all the flows and/or connections in order to take some conclusions about the functionality of the mechanism.

Some of those characteristics can be listed:

- Identification of the flows along the different sections of the network.
- Identification of the packets and flows in the queues.
- Detection of flows throughput when they are crossing trough the queues.
- Discarded packets of the flows in the queue and total lost packets.
- Life duration of the flows.
- Relative position of flows mean throughput and instantaneous respect to the rest of flows.

In particular for the most part of this chapter we focus out the attention to the Cumulative-CS, who showed to be the fastest and less memory consuming of the schemes as was expressed in the chapter 2, in the section 3.5 at page 80 is compared for this simulation scenario the performance of the different counting schemes.

Has to be said that the throughput associated to a flow when it arrives to a queue can be different to the goodput that effectively crosses the queue. The difference throughput-goodput occurs when the queue is completely loaded in different time instants and packets are discarded. Consequently some packets are able to pass through the queue but others are simply lost, thus the throughput and goodput for a given queue can be different.

Part of the study will be focused in the behavior of one particular queue which can be found inside of a router that is crossed by traffic flowing in a single direction. Moreover, that router routes IP traffic that comes from different sources and different inputs of it and that are directed to the same output link; therefore the traffic must pass by the same outgoing queue of the mentioned router.

The figure 3.1 depicts the actual simulation scenario; in it, the computers that conforms the simulation model are separated in to two main groups, one will be the group of servers at the left side of the figure and the other one the group of the clients at the right side.

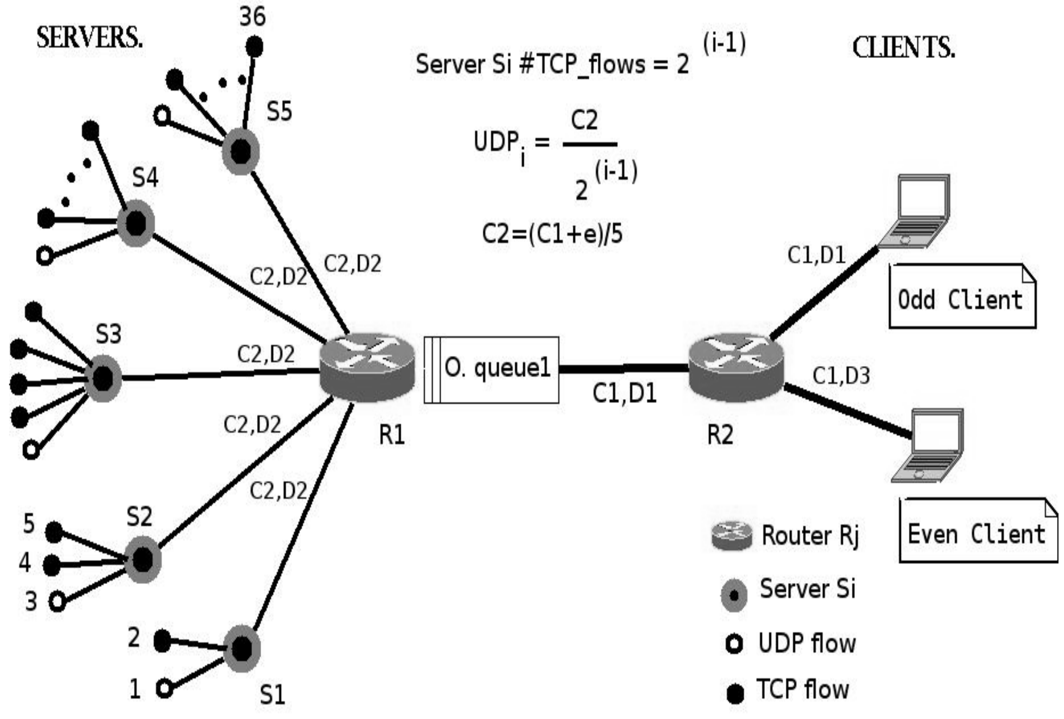

Figure 3.1 Simulation Network

In the same figure can be seen 5 servers that will offer services based on TCP and UDP transport layer protocols, each of the servers is directly connected to the router (who will be named as R1) by mean of different links; these servers offer one service at a time based on UDP and one or more services based on TCP at a time.

The intention is to have different TCP/UDP flows generated from the servers, and that the different servers will offer more TCP based services following a geometric progression based in the number of the server. The progression is $2^{(i-1)}$, where "*i*" is the number of the actual server (for instance, the Server3 can offer $2^{3-1}=4$ TCP based flows services, and Server5 can offer 16 TCP based flows services).

The two clients machines are called Odd-Client and Even-Client, in order that the odd TCP flows in each of the servers will be directed to the Odd-Client and each of the even TCP flows will be directed to the Even-Client. Additionally the odd numerated servers will have their UDP based service connected to the Odd-Client and the same analogy with the even numerated servers. For instance for the server4 which can have 8 TCP based services, the TCP flows 1, 3, 5, 7 will be directed to the Odd-Client and the 2, 4, 6, 8 will be directed to the Even-Client and as this server is even numerated “4” its UDP based service will be directed with the Even-Client.

The TCP flows corresponds to HTTP and FTP service connections, most of them of type FTP in which long files (i.e. 200 MB) are transferred from the servers to the clients. The referred connections are restarted every time they die, in other words after one TCP connection is finished another that will take its place will start in some aleatory based in a uniform distribution function. The switching between FTP and HTTP depends in actual size of the file to transfer, size that is thrown by a exponential distribution at each new TCP flow restart. The idea with such TPC flows is that for most of the time they should be alive and transmitting data. In the table 3.1 is summarized the values and type of distribution used for the TCP flows.

Table 3.1.1 TCP flows principal parameters

Parameter	***Distribution Type***	***Value***
File Length to Transfer	Exponential	Mean: 250 MB.
Connection Restart Time	Uniform	Range: 1-5 seconds

The UDP flows will corresponds to services that offer audio and/or video streaming from the servers to the clients, they are of type Constant Bit Rate (CBR). The rate associated to each of the UDP flows will be the capacity of the link that connects the router R1 with the respective server divided by the number of total flows that go out of the respective server (in the figure 3.1 rate of the UDP flows is indicated as UDP_i and the capacity of the links that connect the router R1 with the servers is indicated as *C2*) . For instance for the Server4 the rate assigned to the UDP flow will be the link capacity divided by nine (9), from where eight (8) are TCP flows and one (1) is UDP; see the table 3.1.3.2 at page 49, it contains the specific bit rates assigned to all the stream services originated by the servers.

Lets call from now on Q*ueue1*, to that outgoing queue of the router R1 that is directly associated with the link that binds the routers R1 and R2. The Queue1 will be the principal point of focus during this simulation study because inside of it will cross the most substantial part of traffic in the simulation and of course because the method presented for HHs detection must be applied in a queue.

3.1.1 Some reasons for the scenario configuration

The separation of the two main group, the “group of servers” and “group of clients” was done with the objective that the clients ask for the services they want from the servers and the servers transmit all that information requested by the clients. Consequently, all the traffic that goes from servers-to-clients will go in the same direction and will cross the same link that connects router R1 and R2. Additionally all the servers-to-clients traffic will cross the same output queue of the router R1 (queue found in the figure 3.1 which is put outside of the router to remark it as a principal point of focus in the simulation study).

Similarly but in contrary direction all the traffic that goes from clients-to-servers will cross the same link that connects routers R1 and R2 and will cross one same output queue for the router R2, traffic mainly constituted by ACKs from TCP, so the reverse direction is not overloaded.

The distribution of even and odd for the flows and Even and Odd for the clients was done with the idea of distributing the load of the servers in a more or less equal mode for the clients, and with the idea that the two clients have traffic from at least four of the servers.

That specific organization of the UDP and TCP outgoing flows from each of the servers was done with the intention that in a situation in where all the flows are alive, each one of them can take approximately the same proportion of bit rate. Obviously, each of the UDP flows will do as much as possible to have their specified bit rate, while the TCP will try to adapt their traffic to the available bandwidth due to its TCP-Reno algorithm which is the one used in the simulation (to see some details of this algorithm see chapter 1.1.3.2 at page 8).

3.1.2 Flows Identification

In the present chapter, at different moments will be necessary to make reference to some specific flows, thus in order to identify them will be presented a numbering system used to name or identify them, this identification will be also called flow-ID.

A good flow-ID for the UDP and TCP flows could be the 5th-tuple composed by (IP- Source, IP-Destination, Source-Port, Destination-Port, IP-Protocol Type) but for simplicity was chosen that each different flow has a different Source-Port; therefore the *Source-Port* number was selected as the flow-ID.

The numbering system for the flows-IDs is explained next: Given a certain flow-ID, in order to know if its an UDP flow and in order to know to which server it belongs to, can be used a simple rule based on dividing the flow-ID by 1000. Like this all the UDP flows have and ID that is a perfect multiple of 1000 (which means that the quotient is an entire number and the reminder is equal to 0) and the Server-ID they belong to, is directly the quotient. For instance the flow with ID= 2000 represents a UDP flow that belongs to the Server2, and the flow-ID=4000 is represents an UDP flow that was generated by the Server4.

To identify if the flow-ID belongs to a TCP flow, the analysis is similar but based in that the reminder is not 0 when dividing the flow-ID by 1000. Thus the Server-ID is given by the quotient but the specific ID of the TCP flow inside all the TCP flows generated by the server is based in the reminder of the division. For instance if the flow-ID is 4003 then when divided by 1000 the quotient is 4 which means it belongs to the Server4 and the reminder is 3 which means is the 3rd TCP flow of that Server4.

3.1.3 Detailed parameter selection

The links that connects the servers with the routers and clients have different capacities and delays; in the figure 3.1 at page 44, the delays are identified as D1, D2, D3 and the link capacities as C1 and C2.

The table 3.1.3.1 shows details of those links, the links can be identified with the two network elements they are connected to (i.e. “Router1 & Router2” refers to the links that connect those routers); ServerX indicates any of the servers like Server1 or Server2, etc.

Table 3.1.3.1 Links Capacities

Link that connects	*Link Capacity*	*Link delay*
ServerX & Router1	2.1 Mbps	10 ms
Router1 & Router2	10 Mbps	0.1 ms
Router2 & Odd-Client	10 Mbps	0.1 ms
Router2 & Even-Client	10 Mbps	0.2 ms

In the simulation, was fixed the **maximum queue length** of any queue to **100 packets.** All the packets that conform any of the flows will have the same size which is equal to 1MB, as a direct consequence the **MCQL** for the Queue1 which serves packets at 10Mbps will be **81.92 ms**, that time fixes the minimum check queue time interval (CQTI) to have the same value in order to avoid over-measuring of the packets inside the queue as commented in the chapter at page.

In the table 3.1.3.2 is shown the assigned bit rate associated to the stream services that use UDP flows for each of the servers and the ideal bit-rates for *each* the TCP flows that come out of a server.

Table 3.1.3.2 Servers UDP and TCP Streams Bit Rate

Server	***UDP flow assigned bit rate***	***TCP flows ideal bit-rate***
1	1.05 Mbps	1.05 Mbps
2	700 kbps	700 kbps
3	420 kbps	420 kbps
4	233 kbps	233 kbps
5	124 kbps	124 kbps

The TCP flows generated from the servers will do their best effort to take their proportion of bandwidth, that will be in the best case the most equal as possible to their UDP counterpart generated by the same server (as was shown in the table). Obviously all the TCP flows cannot have the same bandwidth because there exists a bottle neck formed in Queue1, bottleneck created because the queue1 can serve at maximum 10Mbps and in some time instants the arriving traffic can be up to 10,5Mbps.

Some reasons for the selected parameter values

The different delays assigned to the links that connects the Odd-Client and Even-Client with the router R2 will affect the Round Trip Time (RTT) of the flows associated to links giving different circumstances to the TCP flows that are connected to the Odd or the Even-Client, another reason is that different delays also help to avoid flows synchronization.

For instance if two TCP flows from the same server start their connections with 3ms of difference and the are directed to different clients, as the connection with the Odd-Client will offer a smaller RTT its TCP flow could result benefited and could in fact take a bigger rate than its counterpart. Thus are given different circumstances to the flows, circumstances that are common in the real life connections.

We wanted that the traffic that arrives to the Queue1 could even be in some time instants equal to 10,5Mbps exceeding the traffic it can serve which is 10Mbps. The intention of this traffic excess is that the Queue1 can actually have enough traffic to be able to accumulate packets in the queue, so are created conditions in where the mechanism can effectively be tested.

Is known that the TCP traffic will try to adapt itself to avoid that its packets can be lost, consequently the mean traffic to serve by the queue will not surpass 10Mbps in mean (which is the capacity C1). Besides, there will be enough bursty traffic in different time instants to make that the Queue1 can accumulate packets.

In circumstances in where there is not bursty traffic or that the traffic to serve is so small that the queue associated is in the most of the time empty, the mechanism cannot effectively work to detect the HHs as a consequence that there are not enough packets to be checked in order to take some conclusions. Is in this circumstances in where is convenient to have a threshold engine that can be able to enable or disable the HHs detection mechanism as explained in the chapter 2.

3.2 Flows behavior in the simulation scenario

Before any results study, we will be show which is the theoretical behavior of the flows that come out of the servers in the network and cross the router R1 in order to better understand what are the expectations on the results.

Each of the flows that comes out of the servers can be regrouped into groups that belongs to the server that generate them. For instance the flows that are generated by to the Server3 belong to the group3, the same for the flows generated by the ServerN belong to the groupN.

That mentioned organization of flows with different bit rates (that decrease sequentially when going in increasing direction in the number of the ID for the servers, this is Server1, Server2... ServerN); is done with the intention that can be systematically seen how is the behavior of the groups throughputs. Thus for instance the principal HHs that should be found could belong must likely to the group1 and not to the group5, otherwise the mechanism should not be working properly.

In order to keep a history track of the flows throughput that arrive to the Queue1 during the simulation and in order to avoid to loose any information, was kept track of all the packets that cross the Queue1 not only in the checking instants but also during all the simulation time. Then after every historical check period was saved into a table the mean throughput of each of the flows, an example of that table can be found at table 3.4.1.1.

Finally the behavior of the flows during the simulation respect the expected theoretical value can be seen in the figure 3.2 which shows the incoming mean throughput of the flows that cross Queue1 during all the simulation. In specific in the vertical axis of that figure is shown the mean throughput and in the horizontal axis is shown a number that can help to identify the flow, the plot identified as *Expected* represents the theoretical expected value of the mean throughput for each of the flows. The vertical plotted lines where drown to make easier the distinction of the groups.

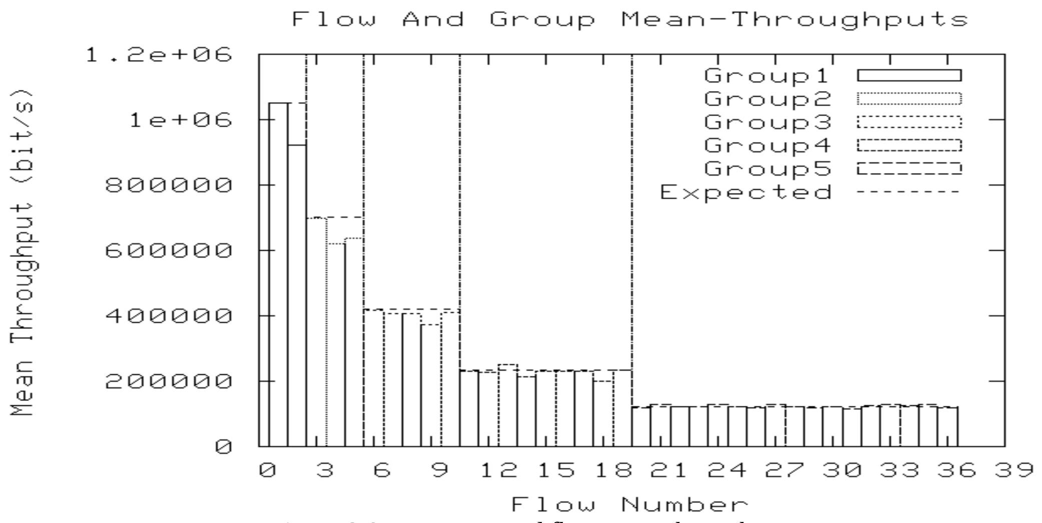

Figure 3.2 Per group and flow mean throughput

See the section 3.2.1 to know how to read the horizontal axis. In the 3.2 graph can be seen that all the UDP flows of each group effectively take as mean throughput numerical values that are really approximate to their expected theoretical bit rate which is shown in the plot with the title “Expected”. Moreover, for the same graph, the rest of the TCP flows for each of the group of flows, have similar values of mean throughput, similar also to their expected theoretical value that in cumulative cannot be bigger than the maximum traffic served by the queue which is 10Mbps.

As a general remark the last figure tell us that the flows are behaving as expected, so the analysis with this flows can be effectively continue.

3.2.1 Reading the horizontal axis of plots

In order to understand how to read the horizontal axis of the figure referred can be taken as example the figure 3.2; should be taken into consideration that the numbering system given to the flows in the graphs is the same as the found in the figure 3.1 at page 44.

In order to identify any desired flow and be able to see its respective value in the vertical axis, the graph should be read in some specific way:

- The flows are separated in groups where each group has a style of plot (i.e. Dotted lines, continuous line, etc.) and represents the flows that come out from one specific server that has the same ID-number of the group.
- The first flow of each group is the UDP flow of the group, therefore the flows that have numbers 1, 3, 6, 11, 20 in the horizontal axis of the graph are UDP.
- All TCP flows of each group keep their original sequence after the UDP flow of the same group; so for instance if is wanted the 3rd TCP flow of the Group4, is found first the UDP flow of the Group4 which has the number 11 in the graph, and then is seen the third flow from it, which has the number 14 in the graph.

- Another help in order to identify the groups are the vertical lines plotted in the figures like the found in the figure 3.2, those vertical lines can help to visually be able to separate the groups and easily detect the UDP flow.

3.3 Parameters to study in the model

In this chapter we will present only one simulation scenario, which is the one that was depicted in the figure 3.1, but will be studied different simulation executions for that scenario. Therefore some parameters will be changed for each of the simulations scenarios that will affect the behavior of the HHs detection model but will not actually modify the behavior of the flows or connections and or services. The behavior of the connections will keep the same criteria, is only the HHs detection model that will actually work differently with those parameters; this with the objective of evaluating the HHs detection model with the different parameters but with flows that keep their same behavior.

Those parameters are:

- NHchks [chapter 2.1].
- CQTI [chapter 2.1].
- N: length of the HHs-List [chapter 2.2].

The *NHchks (Number of consecutive Historical checks)* refers to the number of consecutive historical checks that will be done to the queue during each of the historical check periods. The parameter *CQTI (Check Queue Time Interval),* will determine the granularity of checks in time and as already said cannot be less than the *MCQL(maximum crossing queue latency)* of the associated queue in order to avoid over-measuring [section 2.3.7] of the number of packets in the queue in two consecutive historical checks.

As explained in the chapter 2.2.1 at page 26, the HHs detection model keeps a HHs-List that contains according to the counting scheme the first more relevant N flows during a HCP (Historical Check Period). The idea is that the flows-IDs of the mentioned list should identify the flows that are most likely the Heavy Hitters. To analyze if this mechanism works effectively is kept a table during the execution of all the simulation that saves at every historical check period *the flows in the HHs-List*, where *N* is the maximum number of flows that can be in that list in a certain HCP. Will be referred to this table from now on as the *HHs-Table* [section 3.4.1].

3.4 Mechanism evaluation

3.4.1 The HHs-Table

The HHs-Table is a table that collects different information of some specific flows during the execution of the simulation, its purpose is to assist in the evaluating of the mechanism behavior; this table will help to figure out if the elements of the HHs-List of flows are effectively the HHs.

The mentioned table, saves at every HCP the flow-records inside the HHs-List that can be at maximum N. Moreover, it does not only keeps the flow-ID and the number of packets that have been seen in the queue for each of the flows in the consecutive checks, but also keeps the actual throughput of those flows at every HCP, it also keeps the mean throughput during the flow life, the length of the queue and the m-rank and a-rank of the flow (those names will be explained later in section 3.4.1.1), it also saves the specific parameter values for each of the counting schemes as cumulative number of packets, mean packets, etc.

To avoid any confusion, and as a comment to clarify any doubts, at every HCP the N flows mentioned can be different; that depends in the behavior of the flows of the network configuration at every HCP. Therefore, is possible that the N flows in a HCP are completely different to the N flows in the next HCP, but the idea is to statistically analyze if those flows are effectively the HHs for the majority of the HCPs. Moreover, the idea is also to determine which values of the parameters (NHchks, CQTI) help to have effectively inside the first N flows the HHs.

3.4.1.1 The m-ranks and the a-ranks in the HHs-Table

The *m-rank* is a value given to each of the flows in the simulation execution that indicates which is the position of a flow in respect to others based in their mean throughputs. Thus the flow with m-rank=1 has the highest mean throughput of all the flows, and the flow with m-rank=4 has the forth highest mean throughput of all the flows. Similarly, the *a-rank* indicates the relative position of the flow respect to others based in instantaneous throughputs taken in the last checking instant of a HCP.

The idea is to express with the m-rank the relative *power* of the flows, thus can be seen the relation between the position of the first N flows in the HHs-List and the m-ranks they have. A flow will be considered more *powerful* than another if its mean throughput is bigger.

As a reduced example of the HHs-Table, can be seen the table3.4.1.1 that shows an instance of some possible values that could be taken at one generic simulation for a N=2 and for a **cumulative counting scheme**; this table contains some of the most relevant characteristics observed for the HHs-Tables taken during the simulations.

Table 3.4.1.1 HHs-Table example

Instant of HCP (s)	***Flow-ID***	***Cumulative pckts during HCP***	***a-thrpt (kbps)***	***m-Thrpt (kbps)***	***a-rank***	***m-rank***
110.01	1001	15	984	1000	1	2
110.01	1000	13	952	1048	2	1
110.81	1000	16	1064	1048	1	1
110.81	1001	13	998	1004	2	2
....						
510.18	2000	12	900	701	3	4
510.18	4003	5	251	230	11	12

In the present table “a-thrpt” means actual or present throughput of the flow at the last check of the HCP; “m-thrpt” means the mean throughput of the flow during its life. Additionally, the flow-IDs are shown in a simple numeric way to indicate and identify easily the group number and type of flow (UDP/TCP) of an specific flow in the simulation model. To understand the flow-ID refer to chapter 3.1.2 at page 47.

What is really wanted to be shown inside the illustrated table is:

- The flows shown in the HHs-Table for a given N, do not always have the firsts flow m-ranks, so the 2 flow-IDs of the HHs-List as in the example, does not necessarily means that only the flows of 1st and 2nd m-rank will be selected and shown in the HHs-Table. For instance can be seen that for the HCP time equal to 510.18s, the two flows selected have m-ranks 4th and 12th,, and they where chosen because the number of counted packets for the consecutive checks that involves the related HCP was bigger than for the rest of flows at that instant (remember that for the example is used the cumulative counting scheme).

- The order in which the flows are shown in the table at each HCP, does not necessarily correspond to the order of their m-ranks. For this can be seen the flows at the HCP time equal to 110.01s, in where the order of the flows shown (order given by "Cumulative packets during HCP" column) does not necessarily coincides with the order of the m-ranks. Thus as can be seen, the first flow-ID position is taken by the flow with 2nd m-rank.

Those remarks show that the mechanism does not put at every HCP the flows with lower m-ranks which are supposed to be the HHs, and that flow of 1st m-rank is not necessarily the flow that during a HCP has the greater number of accumulated packets; this could be for some reasons like:

- Existence of a flow which is not really a HH but for a period of time has a reasonable burstiness to put during one or more consecutive HCPs a number packets big enough that the mechanism will select it for those HCPs.

- The flow with 1st m-rank, has a period of low activity during one or more consecutive HCP; ergo even though is the flow that has the highest mean throughput, it could happen that in some periods of time the number of packets it sends is less than the number of packets sent by other flows.

- The CQTI is too long, for instance as long as the life of a HH flow, then the relation between the number of packets cumulated in the consecutive checks will be not related enough and the mechanism will simply not work well.
- The number N of flows to be shown in the HHs-Table is not big enough to give the chance to show the real HHs that maybe are not in those N flows.

In the presented HHs-Table example was tried to show what was actually seen during the different executions of the simulation model. In order to have a general idea of how the selecting HH mechanism really works in some circumstances, and which options and/or parameters should be modified to make it work better.

In short, the mechanism does not always selects correctly the HHs in the N flows that have more cumulated packets in the consecutive checks associated to a HCP (for the case of the Cumulative-CS, although the analysis is the same for the other counting schemes). Indeed, will be studied those parameters of the mechanism as NHchks and CQTI that can help to work in a better way the mechanism as well as the number N of flows inside the HHs-List.

3.4.2 How the model sees the flows

In order to understand how the HHs mechanism behaves, its a good idea to study how the mechanism sees the flows, in other words and more specifically: how are seen the N flows of the HHs-List at every HCP. For the last scope was used the HHs-Table with different values of N and used the data gathered to regroup the information in different plots.

For the next graphs generated from one simulation execution was used when studding the mechanism in the Queue1 the next parameter values: NHchks=3, N=6, CQTI=MCQL with the Cumulative-Cs unless contrary said.

For those given parameters and CS, will be shown how was the mean throughput distribution for each of the different groups of flows in the simulation; being all distributions gathered in one single plot. Remember that a group of flows refers to the flows generated by one specific server, so for instance a flow of Group5 belongs to the flows generated by the Server5.

In the figure 3.3 is expressed the distribution of all the groups and flows mean throughputs during the present simulation execution; throughputs taken at the different historical check periods. In detail, the vertical axis represents the Probability Distribution (named PDF) of the mean throughputs of *each* of the groups of flows taken in each of the HCPs and the horizontal axis shows the mean throughput values.

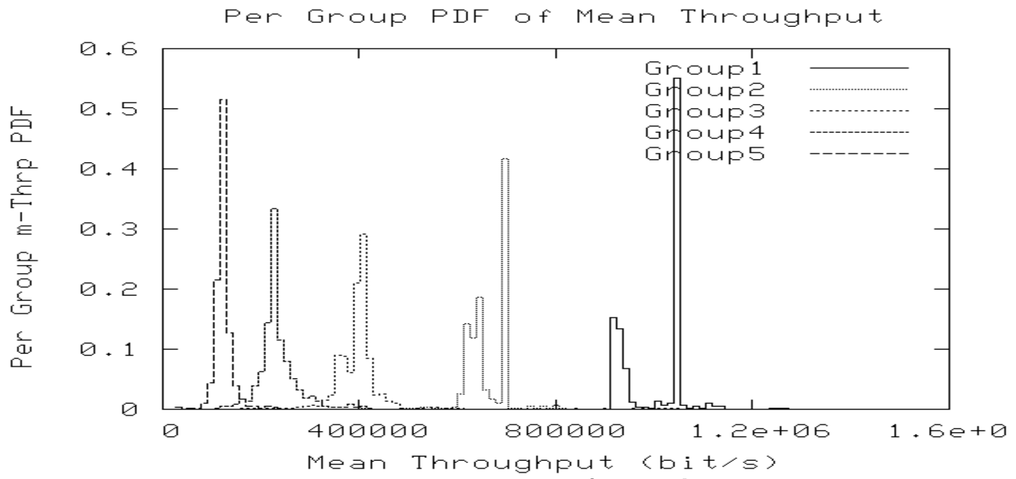

Figure 3.3 Per Group PDF of Mean Thrpt

In that figure can be seen that for each of the groups exists one bar that represents a range of values that stands outs upon the others. For instance in the mean throughputs of the Group2, there is a range bar that represent approximately 44% of the samples present for that group in the HHs-Table, and its value refers exactly to the range of throughput that encloses the bit rate for the UDP flow of the group2. Similarly the same behavior happens with all of the groups, so actually with this graph can help to identified the UDP flows thanks to the condition that those stream flows do not variate their out-coming bit rate as well as do the TCP flows (which is actually right because those are the flows that do not adapt their traffic bit rate to the traffic congestions and to loss of packets). In this manner, those UDP flows are more likely HHs respect their TCP counterparts at least for the first and second group that as can be seen have the higher throughput for their respective group.

Even though the figure 3.3 shows the mean throughputs distribution of the groups of flows and helps to identify the UDP flows from the UDP in this simulation, it does not actually shows how is the occurrence of each of the groups of flows inside the HHs-Table (or in other words, inside the HHs-List during the HCP) and not even the individual occurrence of those flows in the same table.

3.4.2.1 Groups occurrence in the HHs-Table

The figures 3.4 and 3.5 show the occurrence of the groups of flows inside the HHs-Table; in the case of figure 3.4, the value of N is the same as in the other graphics in this section, N=6; and in the case of figure 3.5 the value of N is N=3. In detail in the horizontal axis is shown the ID of the groups for each of the vertical plotted bars, and in the vertical axis is shown the value for the distribution of those groups inside the HHs-Table.

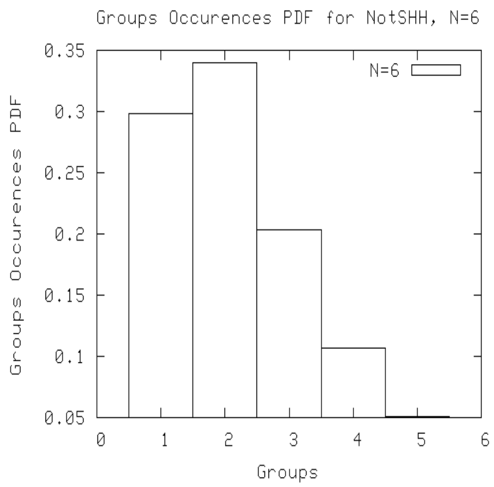

Figure 3.4 PDF of Groups at HHs-Table with N=6

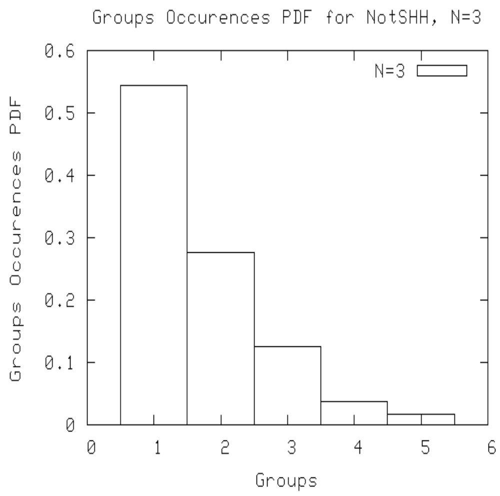

Figure 3.5 PDF of Groups at HHs-Table with N=3

Is important to notice that in the figure 3.4 the bigger proportion of the flows shown in the HHs-Table belongs to the group2 and is approximately 34%. That percentage is actually acceptable because having a HHs-Table that shows 6 flows at a time at each HCP can make that actually in the HHs-Table the flows of the group2 appear more times that the flows of the group1 because the group1 has only 2 flows while the group2 has 3 flows.

In the case of figure 3.5 can be confirmed that the mechanism is actually working in the right direction, this result actually give more presence to the group1 than in the figure 3.4 because N=3 which means that in the HHs-Table will be shown only the 3 flows in the HHs-List at each HCP. Thus is actually expected that the flows of this group have more presence than the others in the HHs-Table. Another detail that the graph says, is that the mechanism is actually in the right way, for this value of N the mechanism gives more relevance as expected in a decreasing manner of the ID of groups, so the flows of the group1 appears more than the flows of group2 and this more than group3 and like this successively; thus for instance is less probable that the mechanism selects in one of his first two flows, one flow that belongs to the group5.

3.4.2.2 Individual occurrence of the flows in the HHs-Table:

When is showed the individual occurrence of each flow with respect to all the flows in the HHs-Table like in the figure 3.6, can be really seen the importance that the mechanism gives to the different flows. In detail on that graph, the plot style of the plotted bars that represents each flow as well as the vertical lines help to identify the different groups; the vertical axis represents the proportion of the number of times this flows appears in the HHs-Table in respect to the other flows or in other words, simply their appearance distribution in the HHs-Table; the horizontal axis helps to identify each individual flow. In order to be able to read the horizontal axis refer to section 3.2.1.

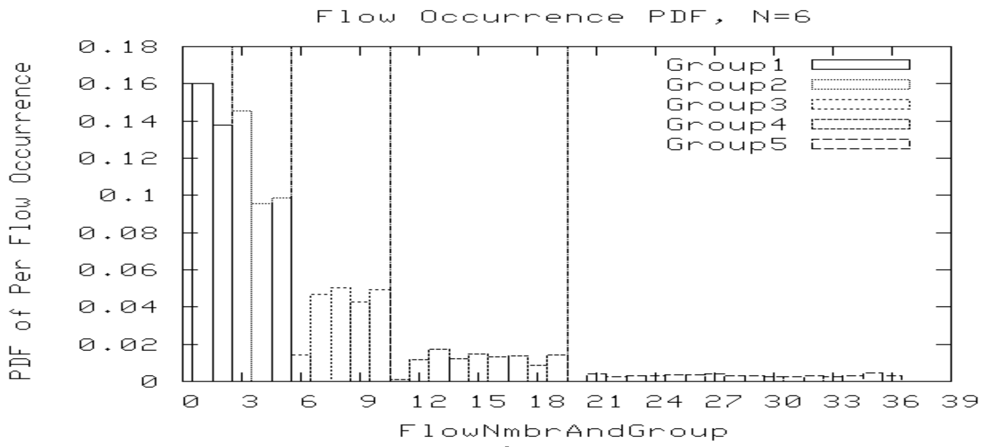

Figure 3.6 Per Flow Occurrence

In the figure 3.6 can be seen that in the HHs-Table is given more importance to the UDP flow of the group1 then to the TCP of the same group1, which is very good because this UDP flow is the most powerful flow in the simulation. A similar analysis can be repeated for the group2, but for the rest of the groups to the UDP flow is given less relevance; a possible reason for this could be that for the rest of the groups, as the UDP flow has not such a high bit rate for their respective channel, the other TCP of the same group can surpass easily the UDP throughput because they have a more bursty bit rate. The characteristic that the TCP can be bursty implies that in some instants of time they can insert more packets in the queue than the UDP counterpart (specially if the UDP has low bit-rate), so the mechanism will give more relevance to the TCP flows in this case.

Even though the figure 3.6 shows which is the importance given to any particular flow in the HHs-Table for the parameters already mentioned, it does not says how it treats specifically the flows that really have a higher mean throughput; because in that graphic is not shown in any way the throughputs of the TCP flows. In order to see these behavior where assigned m-ranks to the flows, value that can be seen also in the HHs-Table.

3.4.2.3 The m-ranks distribution

Would be interesting to see how are distributed the m-ranks [section 3.4.1.1] in the HHs-Table in order to see if effectively in the mechanism the N flows inside the HHs-List are mostly the first N m-ranks.

The figures 3.7 and 3.8 both at page 64, show the distribution and cumulative distribution respectively of the m-ranks in the HHs-Table. The value of N for the mentioned figures is the same as in the other graphics in this section, N=6. The figures 3.9 and 3.10 at page 64, show the distribution and cumulative distribution respectively of the m-ranks in the HHs-Table for a N=2. In detail for all those graphics the horizontal axis gives the number of the m-ranks and the vertical axis gives the value of the distribution or cumulative distribution for the respective case.

M-ranks distribution expectations

In a ideal case in which the detecting mechanism selects only the first N most relevant flows with the first N m-ranks at each HCP, all the distribution flow samples taken in the HHs-Table for a given N should only show the first N m-ranks. Hence, the distribution value for those m-ranks in the HHs-Table should be 1/N for each one, then for the rest of the m-ranks (ranks of value bigger than N) the value of their distribution will be equal to 0 inside the HHs-Table. So for instance for a N=6, the first N m-ranks should have an equal and common value of 0.166 or 16.6% in that distribution and in the case for N=2 the m-ranks should have a common distribution value of 0.5 or 50%.

Another way to look this ideal case can be to look at the cumulative distribution of the m-ranks in the HHs-Table, then if only the flows with the first N m-ranks appear in the HHs-Table after the m-rank of value N should the value of the cumulative distribution should be 1 or 100%.

In the figures 3.7 and 3.7 is plot a curve identified as “ideal” that curve indicates the expectation of the m-ranks for a given N.

M-ranks distribution results

In the graphs of figures 3.7 and 3.9, can be seen that for most of the m-ranks while going in a increasing direction of the m-rank exists a decrease in the value of the distribution.

For the particular case of figure 3.7 in which N=6, the 1st m-rank has a value in the distribution of approximately 16% which is really good and close to the ideal expectations because indicates that for the majority of HCPs the flow with the 1st m-rank is inside the group of the N flows of the HHs-List. Moreover, the 2nd and 3rd m-rank have similar distribution values of around 14% which is good, but for the 4th, 5th and 6th m-ranks the value in the distribution decreases substantially so the rest of distribution is then spread from the 4th m-rank on. The decrease in the distribution from the 4th m-rank means that for a N=6 the mechanism works actually well in identifying the flows with the first 3 m-ranks, detecting their presence in a good part of the HCPs but has less effectiveness for the flows with the m-ranks 4, 5 and 6.

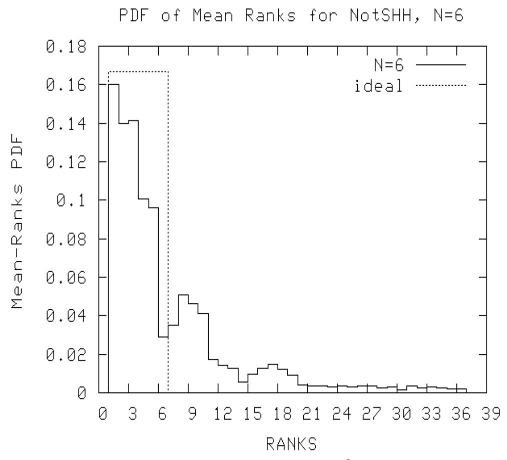

Figure 3.7 m-ranks PDF for N=6

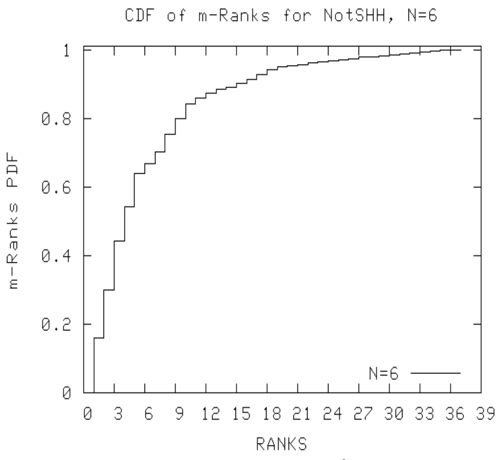

Figure 3.8 m-ranks CDF for N=6

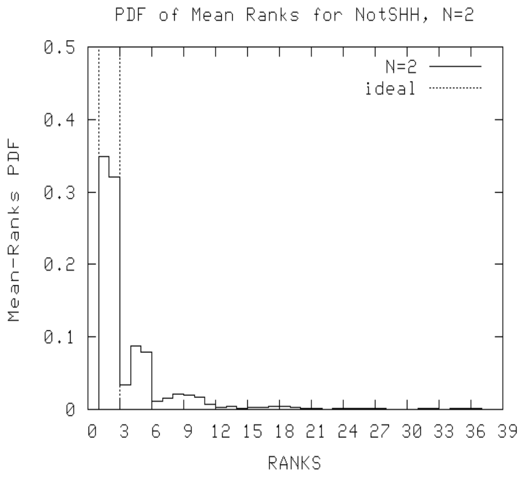

Figure 3.9 m-ranks PDF for N=2

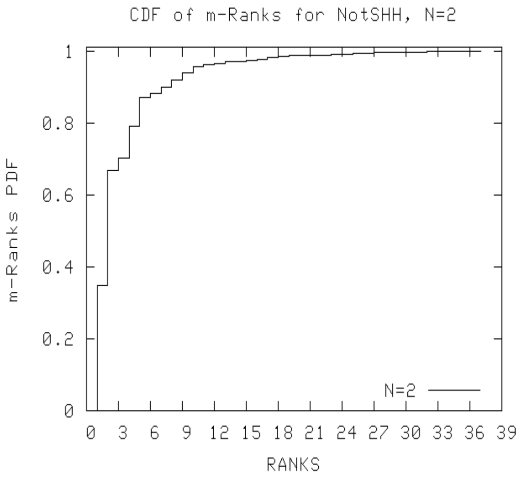

Figure 3.10 m-ranks CDF for N=2

For the figure 3.9 which has N=2, can be seen that for the 1st m-rank is given approximately 35% of the distribution, for 2nd m-rank approximately 33% and for the rest of the m-ranks is spread the distribution. In the same figure, the 1st m-rank appears considerably less than the ideal 50% of the distribution, which indicates that the mechanism for this N is unable to detect the flow with the first m-rank in at lest 30% of the HCPs; that last percentage value because at each HCP is taken N flow samples for the HHs-Table, so the number of HCP is approximately N times less than the total number of flows taken in the HHs-Table during all the simulation execution (is said approximately because not at every HCP exist the N flows, for instance at the beginning of the simulation).

What these graphics show that have been seen for two different values of N, is that the mechanism is able to select in most of the HCPs the flows with the first *L* m-ranks. Where L is is such a number that L<N. In other words, the mechanism can assure for the simulation that the most relevant L flows are inside the first N flows (flow-records) of the HHs-List.

In order to complete the information given in the figures and 3.9 where presented the figures 3.8 and 3.10, that present the cumulative distributions for the m-ranks in the HHs-Table for N=6 and N=2 respectively.

In particular the figure 3.8 says that approximately 70% of the flows taken in the HHs-Table with N=6, belong to the first 6 m-ranks; and the figure 3.10 indicates that approximately 68% of the flows taken in the HHs-Table with N=2 belongs to the first 2 m-ranks, as already said in an ideal case they should have for their value of N already accumulated the 100%.

3.4.2.4 The a-ranks distribution

In the last figures was seen how the mechanism sees the m-ranks, ranks that are supposed to be more important than a-ranks because they say in mean which are the flows that in the mean of the time has the bigger throughputs. Besides, will be interesting to see what happens with the a-ranks which says at every HCP which are the flows that have the highest instantaneous throughputs.

The figures 3.11 and 3.13 show the distribution of the a-ranks in the HHs-Table, for the parameter N of value N=6 and N=2 respectively. In detail for those graphics the horizontal axis gives the number for the ranks and the vertical axis gives the distribution value of the a-ranks in the HHs-Table.

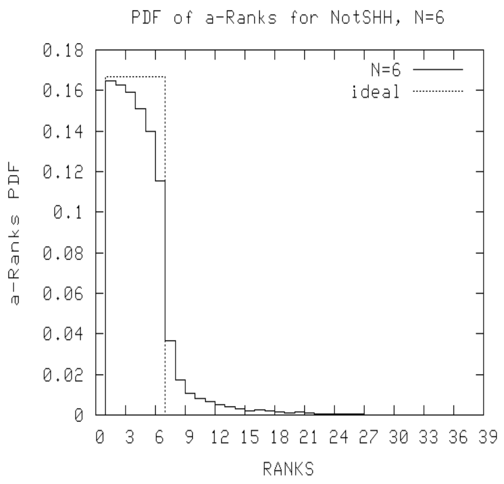

Figure 3.11 PDF a-ranks for N=6

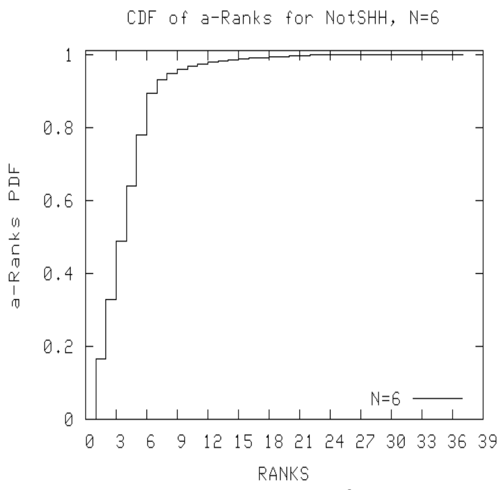

Figure 3.12 CDF a-ranks for N=6

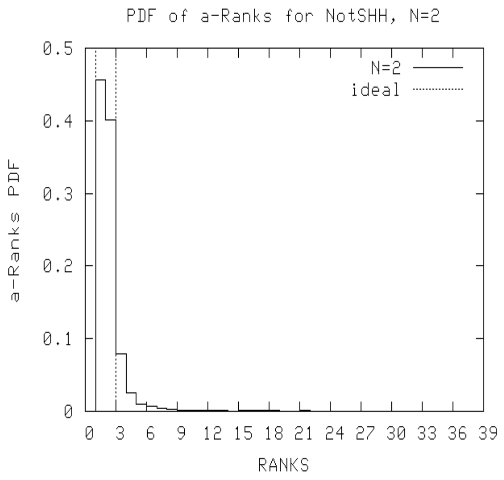

Figure 3.13 PDF a-ranks for N=2

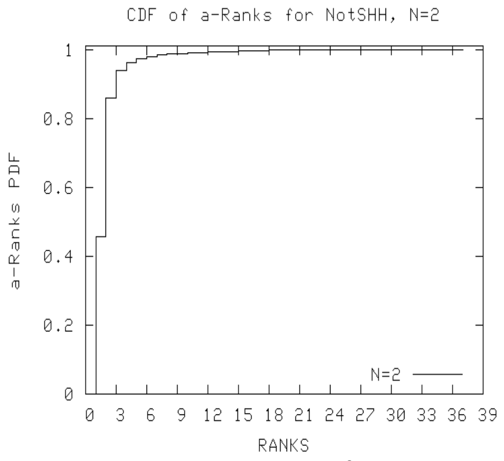

Figure 3.14 CDF a-ranks for N=2

The important remarks for these graphs is that they show that actually the mechanism works substantially better to detect the first N a-ranks than to detect the first N m-ranks.

The figure 3.11 says that for all the its first 6 ranks the distribution is closer of the ideal 16.6% than in the case of m-ranks, and the respective cumulative distribution in figure 3.12 shows that the first 6 ranks cover approximately 90% of the flows taken in the HHs-Table.

Similarly for N=2 the figure 3.13 shows that for for the first two a-ranks their presence is around 40-45% which is much better that in the case of m-ranks and the cumulative distribution as seen in the figure 3.14 for the first two (2) ranks arrives to an approximate of 87%.

Now comparing the results of the figures from to 3.10 and from 3.11 to 3.14 is possible to recognize that the mechanism keeps a better track of the a-ranks than of the m-ranks during the simulation. Better track that is really understandable and expectable, because the delay of each historical check inside a HCP is short enough to detect variation of the flows throughput. In other words, the scan done in the Check-List in order to fulfill the HHs-List at each HCP does not regard historical aspects for longer time than for a HCP, which means that a flow that results to have a bigger actual throughput than another inside a HCP will be more likely included in the HHs-List of the mechanism.

What is wanted to be shown is that the mechanism is also affected by the burstiness of the flows. Thence, flows that can reach considerable high bit rates with respect to the others in small time (shorter than HCP or not to much bigger), are selected by the mechanism.

Another remark that was already mentioned is that for those flows whose mean throughputs are not that different (this is the difference between their mean throughput is less than the bigger of their standard deviations), the mechanism will not be able to distinguish which is really the most powerful or the one who has the highest m-throughput. Besides, the mechanisms can actually detect more frequently the flow that has the highest a-throughput because as already said it follows better the a-ranks, and it will oscillate its selection looking for the flow with lowest a-rank as the different flows oscillate their a-throughput.

3.4.2.5 M-ranks Vs a-ranks

As has been seen, the mechanism behaves better to detect the a-ranks than the m-ranks. The a-ranks can also say the existence of heavy hitters of short duration, but is frequent that flows as TCP take high big rates at different instants while there are adapting to the different condition they are subject to, and that they are not necessarily heavy hitters at long term; that's why the study will be focused from now on the behavior of the mechanism with the m-ranks.

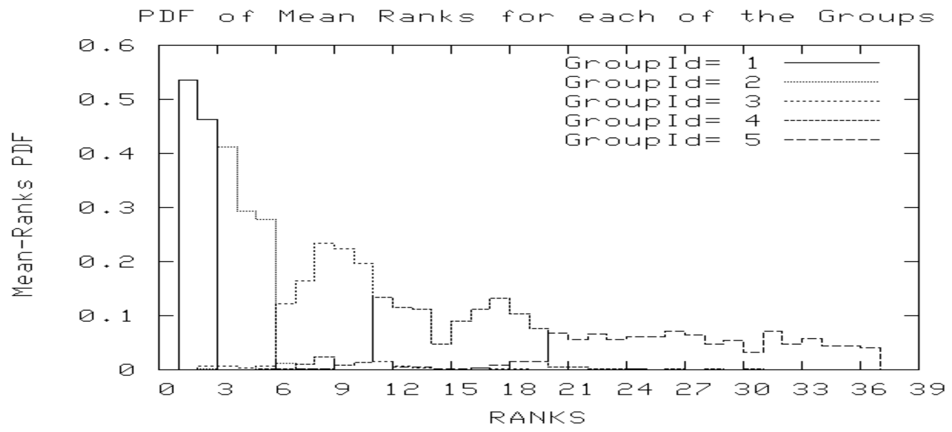

Figure 3.15 m-ranks PDF for the groups

The figures 3.15 and 3.16 show the distribution of the m-ranks and a-ranks respectively for each of the groups at the HHs-Table and the figure 3.17 shows the cumulative distribution of the a-ranks for each of the groups during the simulation.

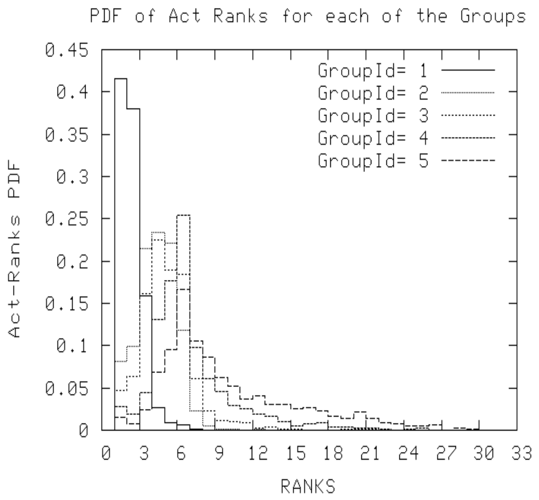

Figure 3.16 PDF of a-ranks for groups

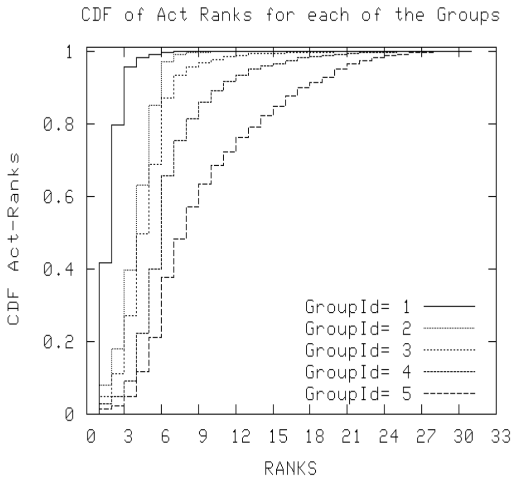

Figure 3.17 CDF of a-ranks for groups

As can be seen in be seen in figure 3.15, the m-ranks can effectively describe the original proposed scenario, in where each group takes the m-ranks they were supposed to. For instance the flows of the first group as have the higher mean throughputs they will take in most of the cases the first two m-ranks even dough they take in less proportion the other ranks.

On the other side the figure 3.16, shows that the distribution of the a-ranks for each of the groups, takes a wider range of ranks than for the figure 3.15. Therefore, this kind of rank cannot actually describe well the behavior of the groups at long term as the m-ranks do, they tent to describe a more granular behavior.

Besides, as can be seen if the figure 3.17 which shows the cumulative distribution, is possible to distinguish the influence of the groups using the a-ranks, because in that figure the cumulative distribution for the a-rank grows faster for the group1 than for the group2 and this faster than group3 and successively. Thus actually, the a-ranks can also describe the behavior of the groups but in a less clear way than the m-ranks.

3.4.3 Selected HH

From this point on, will be studied which is the behavior of the mechanism for detecting what is considered as the principal HH in the simulation, which refers to the UDP flow with ID 1000 and who is generated by the Server1. Moreover, the intention is to see how far the mechanisms can isolate this unique flow in order to do some other studies.

As we have already mentioned in the section3.4.2.3, at each HCP the mechanism does not always selects the flow with the lowest m-rank as the first flow in the HHs-List not even at every HCP; that flow with the lowest m-rank is often inside the groups of N flows in the HHs-List. Thus will be interesting to see how big should be the N parameter to include as much as possible the flow with the lowest m-rank. Therefore, in order to include the flow with the lowest m-rank will be used the Selected-HH (SHH). The ***Selected-HH*** simply refers to select from the N flows inside the HHs-List the flow that have the highest mean throughput. The idea is to see if the distribution for the Selected-HH shows for the most part of the samples that flow selected refers to the flow with the first m-rank.

Remember that N is the maximum number of flows kept in the HHs-Table in one HCP, and also represents the maximum number of flows in the HHs-List at each HCP.

The figure 3.18 shows the distribution for the m-ranks taken by the Selected-HH in each HCP for different values of N (1, 2, 3, 6, 100). In particular, the value N=100 simply means N=Nmax or other words N=100 is put to represent that the HHs-List has a length enough to hold all the flows, it simply have all of them so the Selected-HH in this particular case would represent the flow with the lowest rank possible at each HCP. Consequently, there is not better value possible than the showed for this value of N (equal to 100 in the figure), and it arrives approximately to 99%.

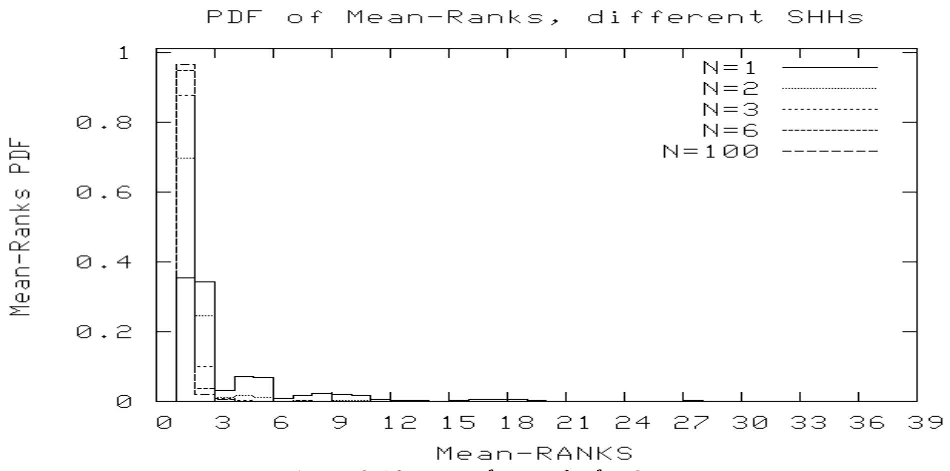

Figure 3.18 PDF of m-ranks for SHH

The figure 3.18 shows also that going in a increasing direction for N from 1 to 6 is increased substantially the distribution percentage for the Selected-HH as the first m-rank. The figure shows also that the increase of N from 6 to 100 for the same rank does not increases its distribution percentage in a accelerated way as for N from 1 to 6; this indicates that in order to ensure that the flow with highest rank is inside the first N flows of the HHs-List could be necessary a value for N that does not necessarily need to be bigger than 6.

In the section 3.4.4 will be studied simultaneously more parameters in order to look to different parameter values that can help to optimize this selection.

The 1st m-rank distribution percentage never arrives to 100% in the SHH

Is approximately 99% the maximum value of HCPs percentage that have the 1st m-rank as the SHH and not the 100% in this simulation for N=Nmax, as a direct consequence of how the code for collecting mean throughput and associating ranks was written.

In the logic of the code, have to pass some number of consecutive HCPs for a flow that has an instantaneous throughput equal to 0 (zero), to be able to determine that its connection has ended and then can be reseted its associated mean throughput and its associated rank; rank which could still be the lowest for some of HCPs. Moreover, the first m-rank is still associated to a dead flow during some HCPs if its mean throughput is bigger that the mean throughput of other flows, and as the mechanism does not chose any flow who have not had packets inside the queue during a HCP that dead flow will be not selected. The mechanism could select the living flow that has the highest mean throughput but that for some HCP is a flow that have the second m-rank because the first m-rank is taken by the dead flow.

For instance, if the TCP connection with flow-ID 1001 has the lowest m-rank for some HCPs and then its connection ends, it could still keep the highest mean throughput for some HCPs and its rank could be still the first. Therefore, the Selected-HH in the next HCPs (in which the flow 1001 is dead but still have the first rank associated) will be the flow whose packets were sampled in the last HCP and that has the highest mean throughput after the dead flow.

Remember that in the simulation each of the TCP connections have a life and when they die, another TCP flow with its same Id and conditions will reborn to substitute it.

3.4.4 Fidelity for the 1st m-rank on the Selected-HH.

Its important to see how the HHs detection mechanism behaves when changing some of its parameters; this section will be focused in studying the ability that the mechanism has to include in the N flows of the HHs-List at each HCP the flow with the 1st m-rank while varying parameters as N, CQTI and NHchks. The purpose, is to determine the best parameter values and conditions for the mechanism to be able to determine the flow with highest mean throughput;

In the graphs to be shown in this section, will be expressed the "Fidelity for the flow with m-rank=1"; this *fidelity* will actually refer to the percentage or proportion of HCPs in where the flow with the 1st m-rank is included in the N flows of the HHs-List for different parameters values. In other words that fidelity will represent the percentage of HCPs in which the Selected-HH posses the 1st m-rank; percentage that will oscillate in a value from 0-99% for the reasons explained in chapter 3.4.3. In short, the fidelity can be express in the formula 8, where the *t* variables represents the simulation time.

$$Fidelity = P\{\ flow\ with\ m-rank=1 \quad \in \quad N\ \ flows\ of\ \ HHs-List \quad \forall\, t\,\} \qquad (8)$$

In the mentioned graphs the parameter "**Dt**" represents directly the parameter CQTI and its presented as a direct multiple of the MCQL. For instance Dt=3 indicates that the parameter CQTI is equal to 3 times the MCQL and Dt=6 indicates that CQTI=6MCQL.

Lets from now on just simply call *fidelity* to the fidelity for the flow with m-rank=1; in the figures 3.19 and 3.20 is expressed how variates the fidelity when varying the values of the parameters NHchks and N and having a fix Dt=1. In particular in the figure 3.20 is plotted by separated the behavior of the mechanism fidelity for each N when increasing the NHchks parameter, on the other side in the figure 3.19 is plotted by separated the behavior of the mechanism fidelity for each NHchks when increasing the N parameter. In particular in this last graph N=100 represents the value of N=Nmax which is putted to indicate that the HHs-List has a length enough that can hold the information of all the flow-records in the Check-List or in other words for simplicity has an infinite length.

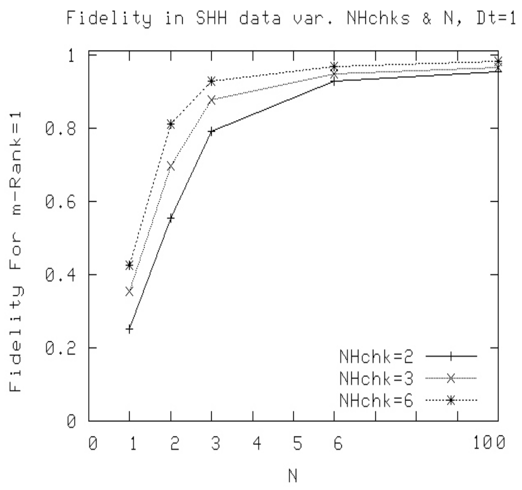

Figure 3.19 Fidelity varying NHchk Vs N

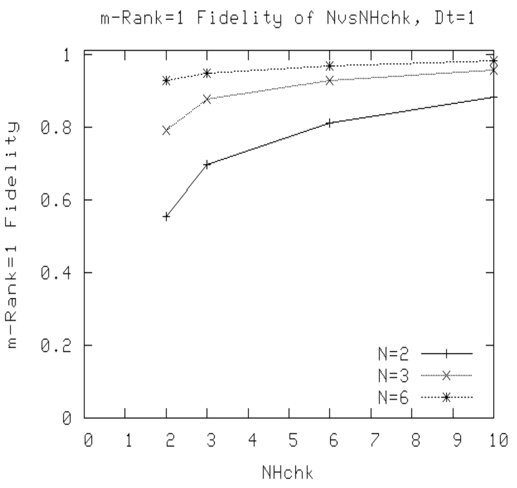

Figure 3.20 Fidelity varying N Vs NHchk

The figure3.19 says as expected that when using a particular Dt and a particular NHchks and increasing the number of N is increased the ability to include in the N flows of the HHs-List the flow with the first m-rank. Moreover, this increase is more relevant for N varying from 1 to 3 than from N varying from 3 to Nmax (N=10 in figure); this last comment becomes more relevant when changing the plot switching to a higher NHchks.

What was wanted to be said is that no matter which is the value for the NHchks or for N the mechanism selects better the flow with the 1st m-rank when increasing N and NHchks; for the case of N the increase is more relevant for small values of N. Therefore, most be done a trade off between keeping a high number for N and the memory necessary for it.

The last two figures actually represent two slides of the figure 3.21 which represents in 3D the fidelity when are varied the parameters NHchks and N for a Dt=1. Remember that the value N=10 actually represents N=Nmax.

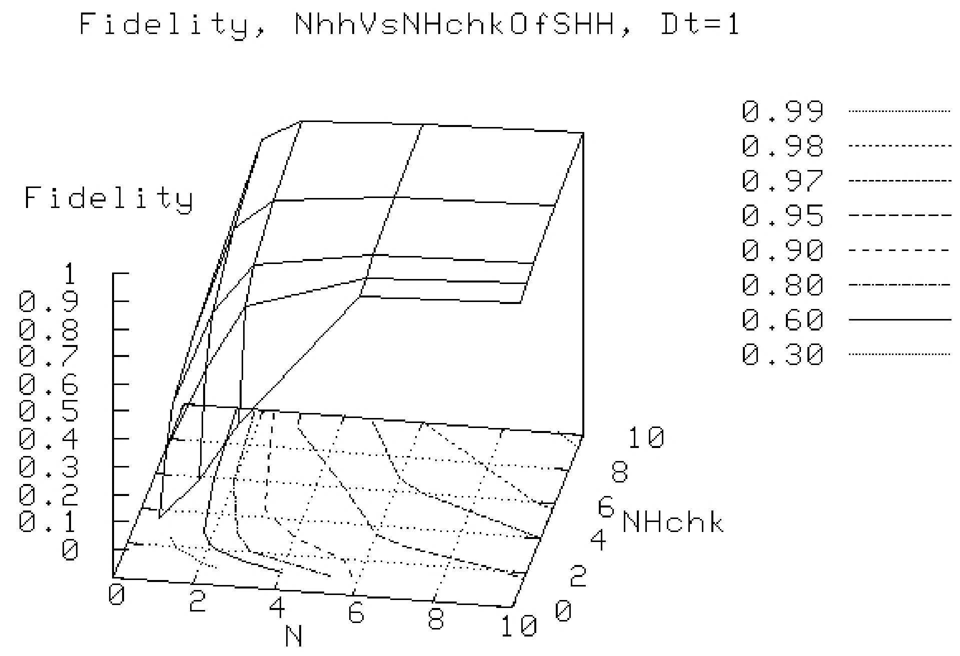

Figure 3.21 3D fidelity varying N and NHchk

The figure 3.20 says analogously that for a fix value of Dt and N, when increasing the NHchks (number of historical checks) is increased the ability to include the first m-rank flows inside the N flows of the HHs-List. Additionally, that increase is not as fast after the NHchks are bigger than 3 and that is not as relevant increase when having a N=6 for any value of the NHchks, as was also seen in graph3.19.

For a fix Dt and N values, increasing the NHchks has the advantages that the queues is checked and info of the flows taken is accumulated in a bigger number of consecutive times. Accordingly, increasing the NHchks parameter has the advantage that the mechanism can be more effective to detect the flows mean throughput rank and help to keep in count the behavior of the flows; but has as drawback that the HHs-List can arrive to multiply its size when increasing the NHchks.

Another aspect is that if the NHchks is too big, the HCP becomes too long and the mechanism takes so much time to give some response about what it shows as HHs. Hence, a trade-off must be done between the advantage of having the best fidelity, the time to give response, the memory to use and processing power needed to handle the Check-List and the HHs-List.

In the figure3.22 is expressed how variates the fidelity when having the N parameter fixed and equal to 3 and are switched values for the NHchks parameter in different plots in the same figure, additionally is let the parameter Dt to be increased from values 1 to 8 for all those plots. Furthermore, as can be seen for each of the plots of NHchks, the fidelity tends to keep constant or with not so much variation and the increase in fidelity comes when switching to a plot that has a bigger NHchk. Thus, can be said that the increase of Dt for a same NHchks and N does not makes any important improvements on the selection of the first m-rank and varying Dt gives non important variations for the fidelity.

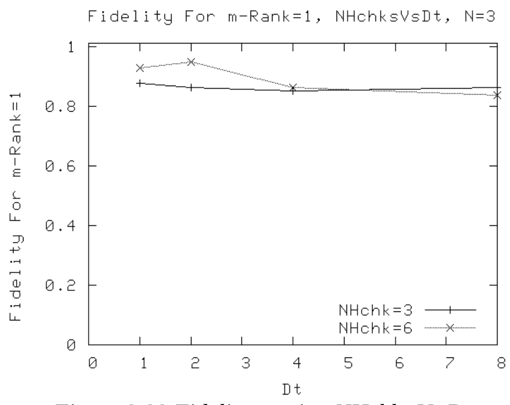

Figure 3.22 Fidelity varying NHchks Vs Dt

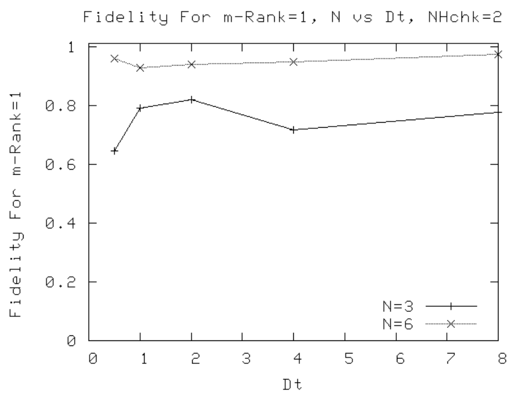

Figure 3.23 Fidelity varying N Vs Dt

In the figure 3.23 is expressed how variates the fidelity when having the parameter NHchk fixed and equal to 2, switching values of the parameter N in different plots in the same graph and letting the parameter Dt to be increased from values 0.5 to 8 for all those plots.

For the same figure, lets focus on a subrange of the values of Dt the range that goes from 1 to 8; for that range as in the figure 3.22 the variation of Dt does not represents an important increase in the fidelity (at least when comparing with the figures in where N and NHchks are variated).

An important remark on the Dt value is that it should not be considerably bigger than the MCQL for two principal reasons: (a) most of the flows of short life duration will be undetected, which is not bad but if Dt arrives to be many minutes the mechanism will only take count of those flows of very long duration but keeping a bad track for them, so simply the mechanism will not work well; (b) while the values of Dt and NHchks grows, the time that the mechanism takes to give response of the flows behavior grows also.

3.4.5 The effect of over measuring in the fidelity

In the figure 3.23 the particular value for Dt equal to 0.5 which is in the horizontal axis, indicates that the consecutive historical checks are done in half of the time in which the queue can serve all the packets if it were completely loaded, in other words CQTI=0.5MCQL. Relation that could introduce problems like recounting a same packet in two consecutive historical checks thus doing over-measuring of the packets.

In order to see what could happen in that case when Dt=0.5, was plotted the fidelity and as can be seen in figure 3.23 for the different plots when changing the value of N, the fidelity value is not necessarily in harmony with the fidelity given in the range from 1 to 8 for Dt. Thus for instance for the plot that has N=3 when Dt passes from 0.5 to 1 the fidelity decreases for then varying in a less abrupt way for the subsequents values of Dt.

3.4.6 The mean m-rank

The *mean m-rank* refers to the mean of the m-ranks <u>in time</u> obtained in all the HCPs during the simulation execution for a given value of N inside the HH-List. This *mean* helps to evaluate how good is the mechanism in order to group the first flow-records that represent the first N more relevant flows (in terms of mean throughput).

The ideal *mean m-rank* (in time) will always increase when increasing the N parameter, because this means that more flows (N) are taken into the HHs-Table at each HCP, so obviously when N is increased the *mean m-rank* will also be increased.

If for instance the mechanism could select precisely at every HCP the first N flows with the first N m-ranks there will be obtained the ideal *mean m-rank* that can be seen in the table 3.4.6.1.

Table 3.4.6.1 Ideal *mean m-ranks*

N	***Mean m-rank***
1	1
2	1.5
3	2
6	3.5
x	$\frac{\sum_{i=1}^{x} i}{x} = \frac{x+1}{2}$

The figures 3.24 and 3.25 show the variation of the *mean m-rank* when varying the parameters N, NHchk and letting a fix value for Dt; this last parameter was not variated because its variation as has been seen before does not affects so much the behavior of the mechanism. In particular in figure 3.24 exists a different plot for each value of NHchk and in the horizontal axis is increased the value for N, on the other side in the figure 3.25 there is a different plot for each value of N and in the horizontal axis is increased the value for NHchk. As a remark for graph3.25 the range of values for N from 7 to 10 are not precise, because the value N=10 represents the value N=Nmax and the values of mean m-rank for N=7, 8, 9 are not real.

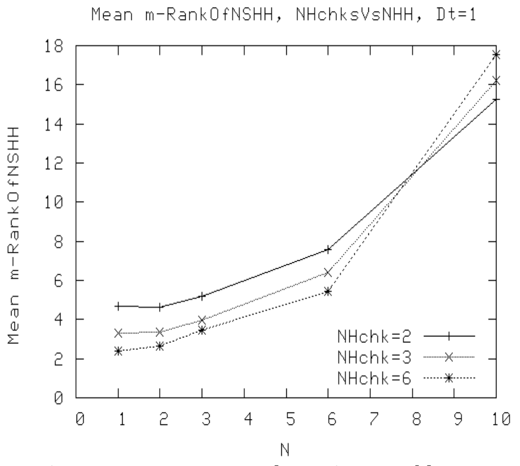

Figure 3.24 Mean m-rank varying NHchk Vs N

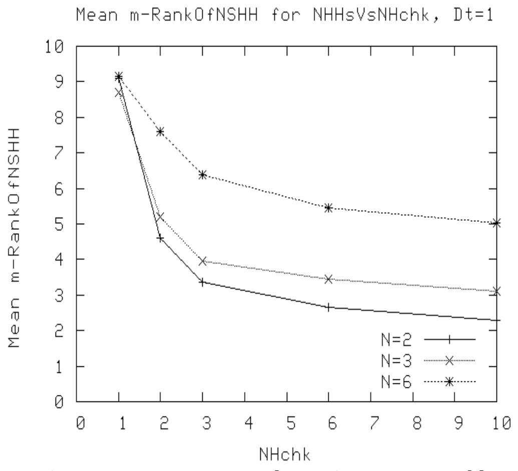

Figure 3.25 Mean m-rank varying N Vs NHchk

In the figure 3.24, can be seen as expected that when increasing the value of N for a plot with fix value of NHchk and Dt the *mean m-rank* gets increased. In particular the *mean m-rank* value that gets closer to its ideal in this graph refers to the the plot that has NHchk=6 in where for instance when N=3 the *mean m-rank* is around a value of 3 which is not that far from its ideal 2 as can be see in the same table.

In the figure 3.25, can be seen that the *mean m-rank* for a fix value of N and Dt gets closer to the ideal when increasing the value of NHchk. Lets focus in the values of the *mean m-rank* when NHchk=10 and see the ideal values of m-ranks for each N in the table 3.4.6.1; for the plot with N=3 the ideal mean m-rank should be 2 and it in the graphic is around 3, this makes for the three plots the minor difference “1” between the mean m-rank reached and the ideal (for the other values of N, N=1 this difference is around 1.2 and for N=6 the difference is around 1.5). Therefore, the best conditions are reached for N=3. To better understand and easily see this analysis will be introduced the concept of *accuracy* in the section 3.4.7.

What can be said from the figures, is that increasing the NHchks parameter, reduces the *mean m-rank* and for the simulation the best values are given for N=3 with a high number of NHchks. Besides, must be remember that the increases of the NHchks parameter will affect the time of response of the mechanism when it grows, as well as the number of flows to handle in the HHs-List; thus so most be done a trade off with these criterias.

3.4.7 Accuracy Vs Fidelity

Even though the *mean m-rank* helps to get an intuition of how good is the mechanism in order to group the first N more relevant flows, is still needed another metric that can directly express this ability, and that is what will be called the accuracy. Thus the **accuracy** express directly the ability of the mechanism to group in the N flow-records of the HHs-List the first N more relevant flows, it is express as: the mean m-rank obtained for a given N ***minus*** the ideal *mean m-rank* for a given N and/or simply:

$$Accuracy=\overline{m-rank}-Ideal\,\overline{m-rank}\ ;\quad for\ a\ given\ N \tag{9}$$

Traditionally when using the word accuracy, is often assumed that the higher the accuracy *value* is the better the better accuracy is reached, but for this case the smallest the accuracy *value* is the better accuracy is reached.

Keeping the fidelity as the ability that the mechanism has to include in its first N flows at each HCP the flow with the first m-rank [refer to formula 8 at page 72]. Will be expressed the relation between the accuracy and fidelity; the general idea is to have the best fidelity with the best accuracy or in other words the less accuracy value possible.

In the figure 3.26 can be seen the relation accuracy-fidelity when is being increased the value of NHchk for a same value of N and Dt; for it look at the arrow that shows the increasing direction for NHchk and/or identify the different plotted points that indicate different values for NHchk; in the same figure is also remembered that the smallest the accuracy value is the best accuracy is got.

In the same graph, the relation accuracy-fidelity gets its best value for NHchk=10, notice that increasing the NHchk is beneficial to both the accuracy and fidelity, the principal cost that most be taken into consideration is that increasing NHchk increases the response time of the HH mechanism as well as the maximum length of the Check-List.

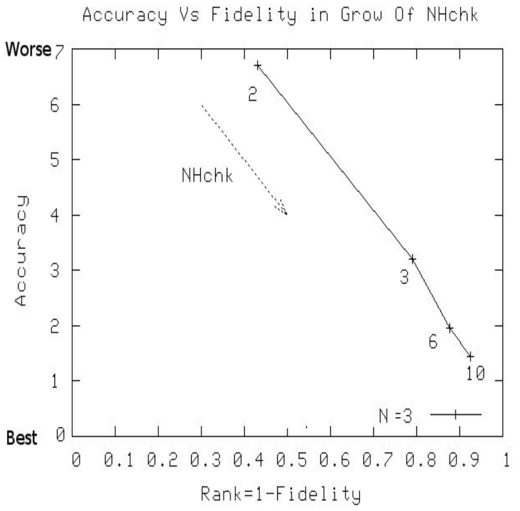

Figure 3.26 Accuracy Vs Fidelity varying NHchk

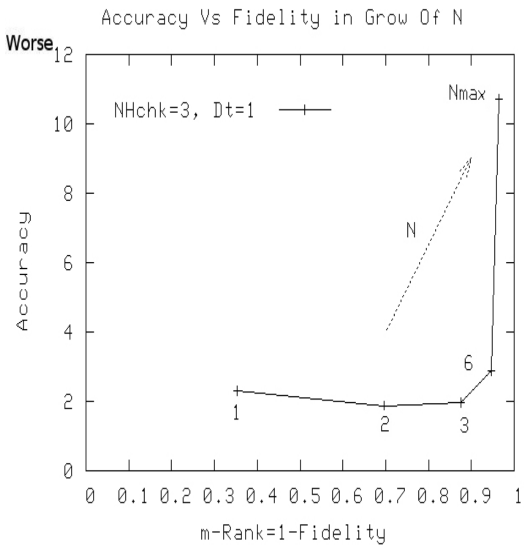

Figure 3.27 Accuracy Vs Fidelity varying N

In the figure 3.27 can be seen on the other side, how is the relation accuracy-fidelity when increasing the value for N and letting fix the NHchk and Dt; for it look at the arrow that shows the increasing direction for N. In the figure, is showed that while N grows the fidelity is enhanced but not necessarily does the accuracy, so a trade-off should be done for the best value of accuracy versus fidelity, that in this case the best choice is for N=3.

3.5 Cumulative, Mean and Vector counting schemes.

All the figures shown in the previous sections of this chapter where created as previously said using the Cumulative-CS, the principal reason is that this counting scheme is the simplest and the best of the CSs proposed as will be seen.

In order to avoid to compare all the previous figures with the ones generated with the Mean-CS will be only shown one figure that involves the most relevant characteristics of the behavior of Mean-Cs and that can be easily compared with those of the Cumulative-CS.

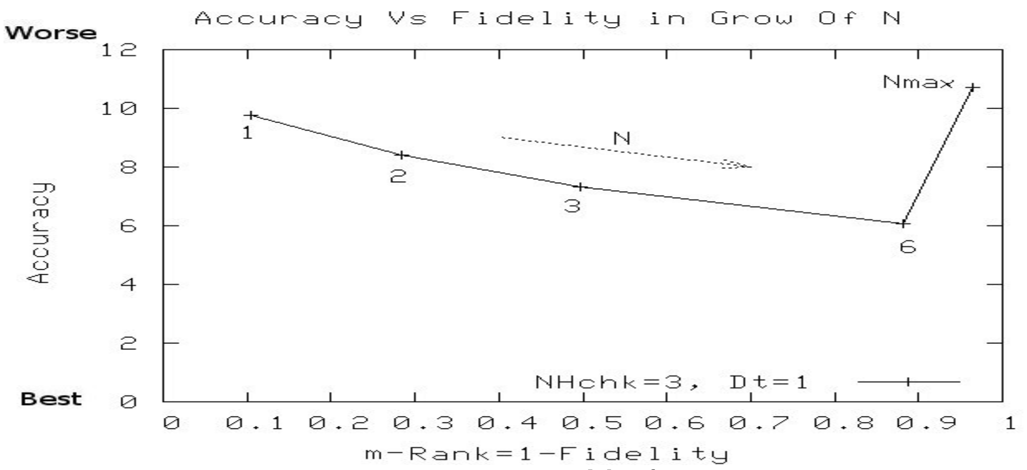

Figure 3.28 Accuracy Vs Fidelity for Mean-CS

The figure 3.28 draws the accuracy Vs the fidelity obtained when using the Mean-CS and increasing N, this figure should be compared with the figure 3.27 (page 80) that also draws the accuracy Vs fidelity in the same conditions but for the Cumulative-CS. In the mentioned figures the behavior of the relation accuracy-fidelity is essentially the same: while N grows the fidelity is enhanced but not necessarily does the accuracy which initially gets better for small values of N but gets worse when N tends to grow.

The important issue to notice and that says that the Cumulative-CS behaves better than the Mean-CS is that for all the values of N the accuracy and also the fidelity have better performance for the Cumulative-CS than for the Mean-CS as can be notice in the figures 3.27 and 3.28. For instance: in the figure 3.27 for N=6 the fidelity is approximately 0.95 and the accuracy is approximately 2.7, while in the figure 3.28 for N=6 the fidelity is approximately 0.9 and the accuracy is approximately 6.

Another reason for not to chose the Mean-CS is that its has a higher complexity than the Cumulative-CS thus it needs to handle floating point numbers and the number of fields used in the flow-records is greater than for the Cumulative-CS, therefore more memory, time and/or processing power is needed in order to evidence the HHs.

By another side the Vector-CS, was not selected as was already discussed in the section 2.2.3, thus the model that is wanted to be used looks to be implemented in some fast element of a network as a router, and not doing any detail monitoring as the Vector-CS can do, so the objective is to reduce the memory, processing power and time to detect the HHs as much as possible but this counting scheme needs more of such resources than the other CSs.

Conclusions

The metric fidelity indicates how good is the mechanisms in order to identify the most relevant of the Heavy Hitters flows, by another side the metric accuracy indicates how good is the mechanism to detect the first N principal heavy hitters.

Overall, the study showed that increasing the Nhchk (number of historical checks) parameter increases the fidelity and improves the accuracy, but must be taken into consideration that increasing the NHchks for a fixed time interval between checks (CQTI) increases the system detection time.

Also, increasing the N (HHs-List length) parameter improves the fidelity but the accuracy tends to be affected when N grows, another issue is that while N is bigger is needed more memory for the HHs-List and more scans need to be done to the Check-List, therefore the time to find the most relevant flows grows (the relevance is given to the flows based in their mean throughputs).

Another result got is that the variation of the CQTI gives not relevant variations in the mechanism performance, must also be noticed that the time interval should be equal or bigger to the time that the queue needs to get empty when it is completely loaded (MCQL), otherwise there is the risk that over-measuring of packets is done. By another side the CQTI should not be too long because it affects the mechanism responsiveness in order to show the most relevant flows; additionally if CQTI is too long many short life flows will not be detected which is actuality not bad but long life flows will not be necessarily well tracked (their evolution).

We have evidenced that the mechanism can follow in a better way the instantaneous variations of throughput that variations of the mean throughput.

The Mean-CS showed that its ability to select the most relevant flows was not better than the offered by the Cumulative-CS. Therefore as the memory, processing power and complexity needed for the Mean-CS was more than for the Cumulative-CS, was therefore left in a second plane the Mean-CS; the same analysis was done for the Vector-CS.

The mechanism does not counts every packet that crosses the queue because the queue is not in every check completely loaded and the time interval between checks can be bigger that the time the queue needs to get empty when it is completely loaded, thats what essentially makes the mechanism viable, because it is not necessary to check every packet that cross the queue and not even to detect all the flows.

An interesting future direction of the current work lies in the use of advance statistical learning techniques, such as Support Vector Machines (SVM), to be coupled to the current HH detection mechanism.

For instance, a possible use would be to feed a SVM with the output of the HH detection mechanism (the HHs-List for each HCP) in the attempt to infer the flow throughput from the information available from the different counting schemes.

Bibliographic References

[1] Berson S., Herzog S.,Zhang L., *Resource ReSerVation Protocol (RSVP)*, RFC2205, September 1997.

[2] Breslau L., Shenker Scott. *Best-Effort versus Reservations: A Simple Comparative Analysis*. ACM SIGCOMM Computer Communications Review, v.28 n.4. October 1998. p. 3-5.

[3] Chen D., Varshney Pramod. *QoS Suport in Wireless Sensor Netorks: A survey*. Proc. of the 2004 International Conference on Wireless Networks (ICWN 2004), Las Vegas, Nevada, USA, June, 2004. p. 1-3

[4] Chuah C., Katz R and Subramanian Lakshminarayanan. *DCAP: Detecting Misbehaving Flows via Collaborative Aggregate Policing*. ACM SIGCOMM Computer Communications Review, v.33 n.5. October 2003, p. 1-2

[5] Cormode G., Korn F., Muthukrishnan S. and Srivastava Divesh. *Finding Hierarchical Heavy Hitters in Data Streams*. Proceedings of the 29th VLDB Conference, 2003. p.1-12

[6] Cuetos De, Philipe. *Streaming de vidéos encodées en couches sur Internet avec adaptation au réseau et au contenu*, (Thesis). --Paris: Ecole Nationale Superieure des Télécommunications, 2003. p. 1-13

[7] Estan Cristian, Varghese George. *New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice*. Proceedings of ACM SIGCOMM 2000. -- San Diego: University of California, 2003 , p 270-313

[8] Hao F, Kodialam M. and Lakshman T.V., *ACCEL-RATE: A Faster Mechanism for Memory Efficient Per-flow Traffic Estimation*. Proceedings of ACM SIGMETRICS 2004. -- Holmdel, New Jersey: Bell Laboratories 2004. p155-165

[9] Kodialam Murali, Lakshman T.V. and Mohanty Shantidev. *Runs bAsed Traffic Estimator: A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation*. Proceedings of INFOCOM 2004. p1808-1817

[10] Krol, Ed. *The Whole Internet User's Guide & Catalog***.** Publisher O'Reilly Media. First Edition. 1992. p. 1-20

[11] Lagoudaskis M., Littman Michael. *Algorithm Selection using Reinforcement Learning*. Proceedings of ICML. p.1-8

[12] Matsuura Yousuke. *Performance Evaluation of TCP Congestion Control Mechanism based on Inline Network Measurement*.(Master Thesis). --.Osaka, Japan. Osaka University. 2006. p.21-24

[13] Rossi, Dario. *A Simulation Study of Web Traffic over DiffServ Networks*. (Master Thesis). --Torino. Politecnico di Torino 2001. p.1-28

Recommendations

[14] , Allman M, Paxson B and Stevens W. *TCP Congestion Control*. RFC2581. April 1999.

[15] Braden R., Clark D., Shenker S., *Integrated Services in the Internet Architecture*, RFC1633, June 1994

[16] Crawley E., Nair R., Rajagopalan B. and Sandick H. *A Framework for QoS-based Routing in the Internet*, RFC2386, August 1998.

Internet

[17] Dictionary.com [online] <http://dictionary.reference.com/>. Word: *Best Effort* [Consultation 2007]

[18] Free On-line Dictionary of Computing [online]. <http://foldoc.org/>. Word: *Data Packet* [Consultation 2007]

[19] Wikibook [online] <http://en.wikibooks.org/> TCP Book: http://en.wikibooks.org/wiki/Computer_Networks/TCP [Consultation 2007]

Bibliography

Breslau L., Shenker Scott. Best-Effort versus Reservations: A Simple Comparative Analysis. ACM SIGCOMM Computer Communications Review, v.28 n.4. October 1998.

Chen D., Varshney Pramod. QoS Suport in Wireless Sensor Netorks: A survey. Proc. of the 2004 International Conference on Wireless Networks (ICWN 2004), Las Vegas, Nevada, USA, June 2004.

Chuah C., Katz R and Subramanian Lakshminarayanan. DCAP: Detecting Misbehaving Flows via Collaborative Aggregate Policing. ACM SIGCOMM Computer Communications Review, v.33 n.5. October 2003.

Cormode G., Korn F., Muthukrishnan S. and Srivastava Divesh. Finding Hierarchical Heavy Hitters in Data Streams. Proceedings of the 29th VLDB Conference, 2003.

Cuetos De, Philipe. Streaming de vidéos encodées en couches sur Internet avec adaptation au réseau et au contenu, (Thesis). --Paris: Ecole Nationale Superieure des Télécommunications, 2003.

Duffield N., Lund C., Sen S., Singh S and Zhan Yin. Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation and Aplications. Proceedings of IMC'04 -- San Diego: University of California, 2004.

Estan Cristian, Varghese George. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. Proceedings of ACM SIGCOMM 2000. -- San Diego: University of California, 2003.

Hao F, Kodialam M. and Lakshman T.V., ACCEL-RATE: A Faster Mechanism for Memory Efficient Per-flow Traffic Estimation. Proceedings of ACM SIGMETRICS 2004. -- Holmdel, New Jersey: Bell Laboratories, 2004.

Hao F., Kodialam M. and Lakshman T.V. and Zhang Hui. Fast, Memory-Efficient Traffic Estimation by Coincidence Counting. Proceedings of INFOCOM, 2005.

Kamiyama Noriaki and Mori Tatsuya. Simple and Accurate Identification of High-Rate Flows by Packet Sampling. Proceedings of INFOCOM, 2006.

Kodialam Murali, Lakshman T.V. and Mohanty Shantidev. Runs bAsed Traffic Estimator: A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation. Proceedings of INFOCOM, 2004.

Krol, Ed. The Whole Internet User's Guide & Catalog**.** Publisher O'Reilly Media. First Edition, 1992.

Kumar V., Stiliais Dimitrios. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Comm.*, vol. 36, no. 5, May 1998.

Kurose J., Ross K. Computer Networking: A Top-Down Aproach Featuring the Internet. Third Edition, Addison Wesley Longman, 2004.

Lagoudaskis M., Littman Michael. Algorithm Selection using Reinforcement Learning. Proceedings of ICML, 2000.

Matsuura Yousuke, Performance Evaluation of TCP Congestion Control Mechanism based on Inline Network Measurement.(Master Thesis). --.Osaka, Japan. Osaka University, 2006.

Rossi, Dario. A Simulation Study of Web Traffic over DiffServ Networks. (Master Thesis). --Torino. Politecnico di Torino, 2001.

Recommendations

RFC768 Postel J., User Datagram Protocol, RFC768, August 1980.

RFC2205 Berson S., Herzog S.,Zhang L., Resource ReSerVation Protocol (RSVP), RFC2205, September 1997.

RFC1633 Braden R., Clark D., Shenker S., Integrated Services in the Internet Architecture, RFC1633, June 1994.

RFC2386 Crawley E., Nair R., Rajagopalan B. and Sandick H. A Framework for QoS-based Routing in the Internet, RFC2386, August 1998.

RFC2474, Baker F, Blake S., Nichols K. Definition of the Differentiated Services Field. RFC2474, December 1998.

RFC2475, Baker F, Blake S., Carlson M. An Architecture for Differentiated Services. RFC2475, December 1998.

RFC2581, Allman M, Paxson B and Stevens W. TCP Congestion Control. RFC2581, April 1999.

RFC2597, Baker F, Heinanen J., Weiss J., Wroclawski J. Assured Forwarding PHB Group. RFC2597, June 1999.

RFC3246 Bennett J, Benson K, Davari S., Firoiu V., Stiliadis D. Expedited Forwarding Per-Hop Behavior. RFC3246, March 2002.

Internet.

Altavista Translator [online]. <http://babelfish.altavista.com/> [Consultation 2008]

Dictionary.com [online]. <http://dictionary.reference.com/> [Consultation 2007]

Free On-line Dictionary of Computing [online]. <http://foldoc.org/> *1993-2007 Denis* [Consultation 2007]

Omnetp Simulator [online]. <http://www.omnetp.org/> [Consultation 2007]

Wikibook [online]. <http://en.wikibooks.org/> [Consultation 2007]

Wikipedia [online]. <http://www.wikipedia.org/> [Consultation 2007]

Glossary

Check-List: indexed list of data-records where the key is the flow-ID, it is used in the HHs detection mechanism and is updated at each historical check inside a HCP , it holds the relevant information of the flows detected in the queue in those checks according to the counting scheme.

Data-Packet: refers to a self contained independent entity of data composed basically by two logical entities, the header or Protocol Control Information *PCI* and the *payload*, the first entity contains the information that can make that the packet can be transmitted in a packet switched network and the second entity refers to the real message to be transmitted. [10]

Data-record or Record: is a data structure that is able to ensembles the information of different types of data.

Datagram: Synonym of data packet usually used in the IP protocol terminology.

DS Code Point, DS Field

Flow: In this context a flow represents a unidirectional stream of packets between a specific source and destination, each defined by a network-layer address and a transport-layer source and destination number.

Flow-set: refers to a group of flows that are being take into consideration at some network node or link.

False Positive: in a system or mechanism that is able to identify if some entity or object represent represents a positive or negative value for the system it self, a false positive will represent that entity which real value is false for the system, but the system fails and recognizes it as an entity with positive value.

False Negative: Is the opposite of a false positive, so in an entity which real value is positive for a given system, but the system fails and recognizes it as an entity with false value.

Group of Consecutive Checks: refers to all the historical checks that are consecutive and that in the time between them the memory associated to the historical checks is not erased. This concept is used only in the present context.

Hash : represents the action of mapping some object (string or concatenation of strings and numbers) by using a Hash function, into another string/number normally smaller in length that is able to identify some characteristic of that object.

HHs-List: list of data-records of maximum length equal to N (fix natural number) that is of type sorted; this list is used in the last historical check of an HCP. In this list is saved the information of the most relevant flows according to the CS during a HCP in form of data record.

Historical Check: makes reference to the moment in which is done the checking of the queue for the HHs mechanism, in other words the moment in where are count the packets for a given flow inside the queue, for accumulating then that information in the Check-List

Historical Checks Period (HCP): is the period of time in which is erased the memory and informion accumulated on the consecutive historical checks, it encloses the time in which a group of consecutive checks have existence. This concept is used only in the present context.

Maximum Crossing Queue Latency (MCQL): is the time taken for a packet of maximum segment size (MSS) from the moment of its arrival till the moment it leaves completely the queue, been this queue such that when the packet arrives it is fully loaded and all the packets are of MSS length, so it is the last packet and all the packets are served with a First In First Out (FIFO) scheme.

Misbehaving Flows: refers to those flows that exceed their stipulation bandwidth limits.

MPLS: A packet switching protocol developed by the IETF that adds a 32-bit label to each packet to improve network efficiency and to enable routers to direct packets along predefined routes in accordance with the required quality of service. Routers make forwarding decisions based purely on the contents of the label. This simplifies the work done by the router, leading to an increase in speed [Reference Dictionary.com].

NHchks: refers to the number of consecutive historical checks that are done to some queue during its historical checks period. This concept is used only in the present context.

Per Hop Behavior (PHB): is the externally observable forwarding behavior and/or treatment that is done to the packets belonging to a PHB class inside a DiffServ network architecture.

Policer: is a device that it’s at the provider side of a network which ensures that the traffic stream generated by the user is inside the limits of some traffic profile otherwise some action can be taken like dropping packages that belongs to the stream.

Shaper: is a device usually in the side of the generating traffic element, that is able to “shape” the generated stream of data altering if necessary its temporal characteristics trying to make it into compliance with some traffic profile.

Socket: Refers to the combination of an IP address and a port number.

Throughput: is the quantity of transmitted data per time unit that is passing over a link or passing over some network node, its measuring unit is often bits per second (bits/s).

Trade-off: can be thought as balancing, it implies a decision to be made in order to lose one quality or aspect of something in return for gaining another quality or aspect having a full comprehension of both the upside and downside of a particular choice.

Latency: in the present context refers to the time that takes for a packet to cross a network, or if specified the time that takes for a packet to cross some portion the network.

APPENDIX A
[Guide for installation and general use of the DropTailQueueExplorable queue-type in the Omnetpp Network Simulator]

École Nationale Supérieure des Télécommunications
INFRES department.

Guide for installation and general use for the DropTailQueueExplorable queue-type in the Omnetpp Network Simulator.

Author:
Miguel Angel Cárdenas Araujo

Paris, December 2007

Table of Contents

A.1 Guide Presentation..3
A.2 Installing Omnetpp...4
A.2.1 Omnetpp-Notes...5
A.2.2 Installing INET..6
A.2.3 INET-Notes..6
A.3 Integrating new files in INET...8
A.3.1 Mandatory contents for INI files...9
A.4 Necessary files for simulating with the DTQE family files............................10
A.5 Parameters to be set in INI files when using DTQE queue-type.....................11
A.5.1 HHs-Table Fields...13
A.5.2 DTQE simulations OUTPUT..15
A.6 Switch Interface between Tkenv & Cmdenv in INET....................................17

A.1 Guide Presentation

In the present guide will be explained how to install OMNETPP and INET, will also be explained how to integrate non-native INET files in INET, and finally will be explained the fields used in the novel queue-type for INET: DropTailQueueExplorable (DTQE) as well as the output fields given by this queue-type.

The mentioned novel queue-type could be used for instance in the Heavy-Hitters detection, could be also used for determining the flows-throughput that cross that queue, and the Output given by that type of queue could also be used in Super Vector Machines for forward uses.

An important remark is that you must already now how to build your Omnetpp simulations as well as with the INET integrated in order to well use the DTQE module. You should also understand how and where to insert the parameters of this modules for instance in the INI files. Thus if your don't now how to use Omnetpp you can read its manual which can be found at: http://www.omnetpp.org/doc/manual/usman.html or at your doc/ folder in your Omnetpp installation folder.

A.2 Installing Omnetpp

In this section are given installation instruction for Omnetpp in Linux from the source files.

The programs TCL-TK, and BLT should be installed and working (i.e. to try tcl use the **tclsh** command and to try tk use the **wish** command). Then the Omnetpp program can be installed:

0. Copy the omnetpp archive to the directory where you want to install it (usually your home directory). Extract the archive using the command:

```
tar zxvf omnetpp.tgz
```

1. Add the corresponding lines (with the right addresses) to your startup file (example for: .bashrc):

export PATH=$PATH:/home/mcardena/simulator/omnetpp-3.3/bin **or** where the bin directory of omnetpp is

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/mcardena/simulator/omnetpp-3.3/lib or where the Ld library is.

export TCL_LIBRARY=/usr/local/lib/tcl8.4 **or** where the tcl lib is

Note: remember do not add spaces between equalities.

2. restart the shell (close and restart the shell window or logout and login again).

3. execute **configure** comand adding the directories of TCL and TK for linux, ie: ./configure --with-tcl=/home/mcardena/simulator/tcl8.4.14/unix/build --with-tk=/home/mcardena/simulator/tk8.4.14/unix/build/ **or** with the directories of TCL and TK for linux

Note: if any advise is given after executing the **configure** command (for instance when setting the TCL_LIBRARY environmental variable to another path), take it into consideration and apply it.

4. use the **make** command: make

(omnetpp is then installed.)

You should now test all samples and check they run correctly. As an example, the dyna example is started by entering the following commands:

```
cd /omnetppHomeDirectory/samples/dyna
./dyna
```

A.2.1 Omnetpp-Notes

The command "*opp_makemake -h*" offers a small help for the command, these is the same for many linux commands, that like this one doesn't have any man "manual page".

In the case you use don't use sh-like shells that do not accept the **export** command, a similar command to use could be: **setenv,** otherwise look at the command that allows to make available the environmental variables and use it in the start-up file as in the example given.
example of use of setenv: "setenv TCL_LIBRARY /usr/local/lib/tcl8.4"

A.2.2 Installing INET

In this section are given installation instruction for INET:

1. Make sure you OMNeT++ installation works. (i.e. try running the samples)

2. Change to the INET directory and edit the inetconfig file
   to make sure it contains the right settings. You'll probably
   need to adjust the environmental variable ROOT to something like this:
   ROOT=/home/michaeilito2/installOmnetpp/INET-20061020 **or** the directory in where omnetpp was installed

3. Type "./makemake" to create the makefiles and the omnetppconfig file.

4. Type "make" to build everything.

5. Add INET/bin directory to the LD_LIBRARY_PATH.

6. If you modify INET and add/remove/rename directories, you'll need to modify
   "makemakefiles", and go to step 4.

That should be it.

Finally, try running the demo simulations. Change into Examples/ and type "./rundemo".

A.2.3 INET-Notes

The most general makefile of INET is in its ./bin directory.

INET is just a omnetpp simulation project like any other that you can create manually, but that has many directories and files. For using it, is enough to create the ned and ini files that will say how the network is configured.

Because INET is a program like any other written in omnetpp, it has just one executable files, which is called INET and is located in the /bin directory of Inet, i.e.:
/home/mcardena/simulator/INET-20061020/bin/**INET**

So a program should call him and must have that address. i.e.:
/home/mcardena/simulator/INET-20061020/bin/INET $*
$* indicates ALL the parameters passed will be included, this is a SHELL

instruction.

Is advisable then, to add the INET executable address to one executable file in order to easily run INET and use the ini files created easily.
For instance:

1. Add the next line to a file named “manualExecutable.exe”:
   $INET_HOME/bin/INET $* # where INET_HOME is an environmental variable that contains the HOME-PATH of INET.
2. run INET with your ini file like this:
   ./manualExecutable.exe -f myIniFile.ini

A good thing about omnetpp is that ned files can even be loaded dinamically, so its no necessary its automatic compilation with nedtool.

Modify INET files is like modifying the source files of any omnetpp program.

The advantage/disadvantage is , the INET is already compiled, so any modification could result in re-making all INET; But that's no necessary because omnetpp allows dynamic loading, so the new files are dynamically compiled, thus it is not necessary the compilation of all the INET.

A.3 Integrating new files in INET

To create simulations based on modifications of INET or simply adding new files that should be integrated with the INET files (for instance the simulations using the file DropTailQueueExplorable.cc) must be created dynamic shared libraries that OMNETPP/INET can use, thus everything will be loaded dynamically.

Therefore must be done the next steps:

1- Go to file omnetppHOME/configure.user and add in it the next line:

LDFLAGS="-Wl,-export-dynamic"

2- type and run **./configure**

3- type and run **make** #like this is recompiled omnetpp

4. Add the next line (with the right addresses) to your startup file (ie. .bashrc)
export INET_HOME=///The path of your INET home////

4.1Then logout and login to the shell.

5. Go to the directory in where the files you want to integrate are, (for instance: in our case the directory where is found the file queueExplorable.cc and run_when_changing_pc.exe)

6. Execute the file run_when_changing_pc.exe, it will generate the objects files and the shared library, in particular: *the shared-library (*.so) will have the same name of your present directory*:
./run_when_changing_pc.exe

If running that file throws errors or if you do not have that file at all, you should run then the next commands manually:

6.1 **rm *.o *.so**

6.2 **opp_makemake -s -f -N -I**dir1 **-I**dir2 **-I**etc

# where the directories dir1, dir2.... are the directories in where can be found the *.h files used in INET, make sure to include all of those directories otherwise the program could not work.

As a general note, this command generates the object and shared libraries as well as the Makefile. In particular the share library will have the same name of your directory, and it accumulates the existent addresses of all the *.h files of INET, like this any new file (*.cc *.h) can include the *.h files of INET (using #include) without the need of writing a complete PATH.

6.3 do the make command: **make**

7 Now you are ready to run your simulation. Make sure that your ini file contain the lines :“**load-libs=...”** and **“preload-ned-files=..”** see the section A.3.1 for more details about those lines.

Thus Run your INET with your ini file as normal. If you want see INET NOTES section for more details.

Note: if you modify the *.cc *. h files you need to do a make command every time you modify them: **make.**

A.3.1 Mandatory contents for INI files

There are some mandatory contents that any *.ini file that makes reference to an non-native integrated file in INET (i.e. DropTailQueueExplorable.cc) should have. Those are the next two lines:

load-libs="mylib"

#where mylib is the name of the shared library created. Note: when running the *run_when_changing_pc.exe* executable file, the name of your shared library will be the same name of your present directory.

preload-ned-files = ./MyNed1.ned ./MYned2.ned @/INET-HOME/nedfiles.lst

# where the **nedfiles.lst** file is a file normally found in the INET_HOME directory that contains the list of *.ned files used by INET by default, and **MyNed1.ned MyNed2.ned** are simply ned files in your present directory that you want to make reference to. That particular line allows the pre-load of the ned files that will be used in simulation

A.4 Necessary files for simulating with the DTQE family files

In this section are introduced mandatory and non mandatory files needed in order to be able to use the queueType “DropTailQueueExplorable”

Mandatory files:

1. queueExplorable.h
2. queueExplorable.cc
3. DropTailQueueExplorable.h
4. DropTailQueueExplorable.cc
5. DropTailQueueExplorable.ned
6. manualExecutable.exe **or** executable file that includes the path to the INET executable.

Mandatory files but that can be different in each simulation:

1. testQueueExplorable.ned **or** file that have the ned configuration for your simulation.
2. testQueueExplorable.ini **or** INI file to be used in the simulation.

Not mandatory but recommendable file:

1. run_when_changing_pc.exe

    # file that erases the old *.so *.o files and executes the opp_makemake -s -f -N -Idir1 -Idir2 -Ietc command, which generates new object and shared libraries as well as the Makefile. In order to make this file work correctly, the environmental variable INET_HOME must be set.

A.5 Parameters to be set in INI files when using DTQE queue-type

First of all, must be remember that the *DropTailQueueExplorable (DTQE)* family files where created to generate a type of queue that is compatible and usable in any INET module that uses the standard INET queue-types.

So simply use in the INI files this new queue-type like any other but just with the parameters to be introduced and explained next.

Must also be said that as the *DropTailQueueExplorable* files are non-native INET files, they must be integrated in INET, follow instruction in section A.3 in order to integrate them. Moreover all the INI files that use the *DropTailQueueExplorable* queue-Type must include the lines explained in section A.3.1

Parameters:

queueType = "DropTailQueueExplorable".
#that is the value that specifies this queueType, when using it all the other parameters to be explained can be used.

frameCapacity= refers to the maximum queue length, after it the packets will be dropped.

HCP= refers to Historical Checks Period, is the period in which are done all the historical checks, and the information of the list is accumulated and scan to then print the HHs-List in the HHs-Tables

append_SVMtablesOfDiffSimulations: is for appending the HHs-Tables of consecutive simulations, executed one after the other. For instance could be used when doing different RUNS (same simulation but using different seed for the Random Number Generator).
If value = 0, means that is not append the HHs-Table with old ones; this is, if there is a old HHs-TABLE with the same name of a New one it will be erased and created the new one.
If value = 1, means that the new HHs-Tables will be appended with the new ones; this is, if a new HHs-table to be generated has the same name of an old one, it will simply continue writing the values in the old HHs-table file.
NOTE: the HHs-TABLES are simply appended, the time and all other values restarted when new simulation are done, the objective of this appending is to have more statistics for a same scenario, simply looking at it with different values.

NHH: indicates the maximum length of records printed in the HHs-Table that is has not the Selected flow in a certain time instant, and or the maximum length

NHchks: indicates the number of times that will be done the Historical Checks to the queue inside the internal_throuput_time_interval. The separation of those checks is simply given by: HCP/NHchks

sortingType: Specifies the way the flows are sorted and/or the way is given relevance to the flows. With other words, it specifies which is the parameter gotten from the flows that decides which flow is more important than other and should take a relevant position, the relevance is given by higher value, so the flows with higher values of the selected parameter will be the most relevants.

The sorting types are:

1. actNumPkg_inQueue
2. oldNumPkg_inQueue
3. actNumPkg_inQueue + oldNumPkg_inQueue
4. hist_pkts_mean
5. hist_pkts_mean – hist_pkts_stdDev
6. hist_pkts_sum
7. meanFlowThrpt
8. actFlowThrouput

For instance the sortingType=6 that refers to the parameter *hist_pkts_sum*, gives relevance to the flow with higher value for that parameter, so a flow with *hist_pkts_sum=6* will be more important than another flow with *hist_pkts_sum=5.*

In the case the value of that parameter is the same for two flows the relevance then will be given in the next order (using this other parameters to compare): actNumPkg_inQueue, actFlowThrouput, oldNumPkg_inQueue,.oldFlowThrouput.

Example of what could be written in a INI file that uses this queueType:
i.e.

```
Network.Router.ppp[0].queueType = "DropTailQueueExplorable"
Network.Router.ppp[0].sortingType=1          #is used actNumPkg_inQueue as sortingType
Network.Router.ppp[0].NHchks=3
Network.Router.ppp[0].NHH=2
Network.Router.ppp[0].frameCapacity=100
Network.Router.ppp[0].HCP= 0.08192
```

Network.Router.ppp[0].append_SVMtablesOfDiffSimulations= 0

A.5.1 HHs-Table Fields

In this section are introduced the fields that can be found in the output of the HHs-Tables which is the name given to the output files generated when using the DropTailQueueExplorable queue-type. First of all will be introduced the concept of rank that will be used by the fields and then the fields will be explained.

rank: is the relative position of a certain flow respect to the other flows following some relevance, the relevance is given to the flows who have bigger *meanFlowThrpt* in the case for the *meanThRank* and bigger *actFlowThrpt* in the case of *actFlThRank*. The value of rank=1 is given to the most relevant flow according to the way the relevance was given.

IdSourceP: refers to the source port of the flow. It is a deprecated field that initially was used to Identify the flows in the case they had all different source ports.

simTime: refers to the actual simulation time, in where the data showed in the present line for the present flow was generated. Or in other words the instant of time in where was end the actual HCP.

actPcksQueue: is the number of packets sampled for a given flow in the last queue check of the present HCP.

oldPkgsQueue: is the number of packets sampled for a given flow in the last queue check of the last HCP (the HCP before the present one).

actFlowThrpt: is the throughput of the present flow when crossing the queue calculated only during the present HCP.

oldFlowThrpt: is the throughput of the present flow when crossing the queue calculated only during the last HCP.

meanFlowThrpt: is the mean throughput of the present flow when crossing the queue, calculated from its detected existence (in the queue) till the end of the simulation or only during the life of that flow. That option depends on the *isResetOfFlowMeanThr_enable* parameter. Thus if that *isResetOfFlowMeanThr_enable* is enabled (=1) the mean throughput of the flow is calculated from its creation till the moment flow does not sends more packets during a *minFlowDeathTimeOut*, thus if the flow sends packets again after that period of time the *meanFlowThrpt* will start again to be calculated. If the same parameter (*isResetOfFlowMeanThr_enable*) is disabled (=0), the *meanFlowThrpt* for the flow is calculated from its creation till the end of the simulation.

n: represents the sample number used to calculate the *meanFlowThrpt* for each flow, it gets incremented by one unit every HCP. As the *meanFlowThrpt* its value depends in the *isResetOfFlowMeanThr_enable* parameter. Thus if that *isResetOfFlowMeanThr_enable* is enabled (=1) the number of the sample grows from the moment of the flow creation till the moment flow does not sends more packets during a *minFlowDeathTimeOut*, thus if the flow sends packets again after that period of time the sample number (n) will start again from 1 (is reset). If the same parameter (*isResetOfFlowMeanThr_enable*) is disabled (=0), the *the sample number (n)* for the flow is gets incremented from its creation till the end of the simulation.

meanThRank: is the rank given to the flow according to its *meanFlowThrpt*.

actFlThRank: is the rank given to the flow according to its *actFlowThrpt*.

actQueueLength: is the length of the queue in the last check of the present HCP.

oldQueueLength: is the length of the queue in the last check of the last HCP.

actvInQFlNmbr: indicates the number of flows present in the queue in the last check of the present HCP.

totalThrpt: represents the total throughput that incomes to the queue in the last check of the present HCP.

histPksMean: represents the mean of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The mean is calculated taking as samples the total number of packets for each of the checks involved.

histPksStdDev: represents the standar deviation of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The StdDev is calculated taking as samples the total number of packets for each of the checks involved.

histCtr: is used for the histPksMean and histPv, represents the current number of samples in where that mean is calculated. In other words the number of historical checks in the last HCP that where used to calculate the histPksMean and histPv

histPv: is the pseudo-variance of the packets for a given flow during the present HCP from the first check (inside this HCP) in where was detected till the end of the present HCP. The histPv is calculated taking as samples the total number of packets for each of the checks involved.

histPksSum: is the cumulative of the packets sampled in each check inside a HCP for a given flow.

flowID: refers to the fifth-tuple that identifies the flow. This is IP source address, IP destination Adress, Source Port, Destination Port, IP transport protocol type. As note the IP transport protocol type (=6) refers to a TCP flow and the (=17) refers to an UDP flow. An example of the fifth-tupla is : 10.0.0.1:1000@10.0.0.6:11000@17

A.5.2 DTQE simulations OUTPUT

There are three two types of output the *DropTailQueueExplarable* queue-type generates, one is in the files omnetpp.sca and the other are HHs-Tables generated.

Case of omnetpp.sca:

Will be output some scalars that indicates characteristics of the *DropTailQueueExplarable* queue-type. The name of those scalars indicate directly the property they are making reference to:

_period_drops_mean
_period_drops_variance
_period_drops_StandardDeviation
_period_drops_min
_period_drops_max
_medium_queue_length_mean
_medium_queue_length_variance
_medium_queue_length_StandardDeviation
_medium_queue_length_min
_medium_queue_length_max

Note: the word *period* in the above scalars refers to Historical Check Period (HCP).

Other scalars, common in all the queue-types (Not only in DropTailQueueExplorable):
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"
"packets dropped by queue"
"packets received by queue"

Case of HHs-Tables:
There are three types of output for HHs-Tables

queueHistoric: this prints at each simulation HCP at maximum the first *NHH* flows that are the most relevant according to the *sortingType.*

queueHistoric_Selected_MAX_ACTFlowThr: this prints at each simulation HCP the flow that has the maximum *actFlowThrpt* from the first *NHH* flows that are the most relevant according to the *sortingType.*

queueHistoric_Selected_MAX_MEANthrp: this prints at each simulation HCP the flow that has the maximum *actFlowThrpt* from the first *NHH* flows that are the most relevant according to the *sortingType*.

An example of the file output name could be:
IP_TCP_UDP_Network.RouterVector[0].ppp[5].queueHistoric_HH=6.dat

That file name indicates that the DropTailQueueExplorable (DTQE) used is found in the network "IP_TCP_UDP_Network" inside the router "RouterVector[0]" in the network interface "ppp[5]" and that uses as NHH the value of 6 (six) and that in its contents is printed the at each HCP at maximum the first *NHH* flows that are the most relevant according to the *sortingType.*

A.6 Switch Interface between Tkenv & Cmdenv in INET

Sometimes is convenient to deactivate the graphic interface (Tkenv) of Omnetpp in order to make run your simulations smoothly and faster, or when what you care is to obtain the output of a simulation and forget about the graphic interface. There's when becomes important to have a simple command line interface like the offerd by Cmdenv.

In order to swithc your interface you must do the next steps:

1- INET has to be already installed and working perfectly.

2- Go to the $INET-HOME/bin/ directory and modify the “Makefile” uncommenting the cmdenv and commenting the tkenv like the next (or the contrary if its your interest):

```
USERIF_LIBS=$(CMDENV_LIBS)
#USERIF_LIBS=$(TKENV_LIBS)
```

3- Save the file and Type: make

So then will be recompiled INET and created the INET executable that will run now in Cmdenv mode, so all the simulation of INET as use this executable they will be directly in Cmdenv.
To return to Tkenv just do the contrary, in commenting and uncommenting and remember to do a “make”.

APPENDIX B
[Resumen en español]

Miguel Ángel Cárdenas Araujo

PROPUESTA Y ESTUDIO SIMULATIVO DE UN NUEVO MECANISMO PARA LA DETECCIÓN DE FLUJOS HEAVY HITTERS.

Tutor Académico: Andrea Bianco. Tutor de Pasantías: Dario Rossi. Tesís. Caracas. U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Ingeniero Electricista. Opción: Comunicaciones. Institución: École Nationale Supérieure des Télécommunications. 2008. 92h+anexos

Palabras Claves: Heavy Hitter (HH); Colas; Simulación de Redes de Telecomunicaciones; Esquemas de conteo.

Resumen. En la presente tesís es propuesto un nuevo mecanismo con tres (3) variaciones (esquemas de conteo) usados para la detección de flujos “Heavy Hitters” (flujos cuya carga en un conjunto de flujos es muy relevante). El mecanismo ha de ser aplicado en las colas de salida de los routers y consiste en evidenciar cada cierto intervalo de tiempo (llamado HCP) lo flujos que él mismo considera como HHs aplicando un esquema de conteo; estos esquemas son: (1) Acumulativo, (2) Medio, (3) Vector. Al interno de un HCP se realizan un número fijo de controles a la cola, los cuales están separados por otro intervalo de tiempo. En cada control se cuenta el numero de paquetes para cada flujo en la cola y se relaciona esa información con la de los otros controles aplicando el debido esquema de conteo; es de notar que la información acumulada sola ha una duración de un HCP. El mecanismo es evaluado en un escenario simulado donde existen flujos TCP y UDP con una variedad de *bit rates*. En particular se denota la atención al esquema acumulativo el cual en estudios hechos demostró ser el más eficiente en el uso de la memoria así como el más rápido. En particular, se llegó a la conclusión que al aumentar el número el número de controles en la cola mejora la selección de los HHs por el mecanismo, pero que el incremento del intervalo de tiempo entre los mismos controles no ofrece una mejora considerable en la selección. Fueron comparados los resultados obtenidos por los diferentes esquemas, y se notó que el esquema acumulativo ofrecía una mejor detección de los flujos HH