Universidade de Aveiro
2008

Departamento de Electrónica, Telecomunicações e Informática

Hélder Manuel Lima Veiga

Sistema Distribuído de Monitorização de Tráfego com uma Arquitectura *Peer-to-Peer*

Distributed Traffic Measurement System with a Peer-to-Peer Architecture

Universidade de Aveiro
2008

Departamento de Electrónica, Telecomunicações e Informática

Hélder Manuel Lima Veiga

Sistema Distribuído de Monitorização de Tráfego com uma Arquitectura *Peer-to-Peer*

Distributed Traffic Measurement System with a Peer-to-Peer Architecture

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. Rui Valadas, Professor associado com agregação, e do Dr. Paulo Salvador, Professor auxiliar convidado, ambos professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho aos meus pais e ao meu irmão agradecendo a amizade, o apoio, a dedicação e a maravilhosa família que me presentearam.

o júri

presidente

Prof. Dr. Joaquim Arnaldo Carvalho Martins
professor catedrático da Universidade de Aveiro

Prof. Dr. Manuel Alberto Pereira Ricardo
professor associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. Rui Jorge Morais Tomaz Valadas
professor associado com agregação da Universidade de Aveiro

Prof. Dr. Paulo Jorge Salvador Serra Ferreira
professor auxiliar convidado da Universidade de Aveiro

agradecimentos
acknowledgements

First of all I express my sincere thanks to my supervisors, Dr. Rui Valadas and Dr. Paulo Salvador for their support, patience, guidance and assistance throughout this research. A considerable part of the DTMS-P2P system is product of their fertile mind and its transition from idea to reality could not have happened without their guidance throughout our many intriguing and inspiring conversations. Moreover, without their suggestions and provoking ideas, this dissertation would not have become as extensive as it has. Also thank you to Dr. António Nogueira for his help and support.

With great pleasure and pride, I would like to thank the Institute of Telecommunications – Aveiro University Pole (IT), for their sponsorship in pursuing this graduate program.

I would like to express a very special thank you to my parents and to my brother without whose support and encouragement this work would not have been possible. Another special thank you goes to my lovely girlfriend, Vanessa Leite, for her love, friendship, support, understanding and patience. Finally I would like to thank all my friends for their friendship and support.

I would like to dedicate this thesis to my parents and to my brother. I owe them a debt that I have no hope of ever repaying.

Aveiro, July 2008

Hélder Veiga

palavras-chave

Monitorização de tráfego, *peer-to-peer*, controlo distribuído, armazenamento distribuído, acesso distribuído, ponto de medição.

resumo

As características do tráfego na Internet são cada vez mais complexas devido à crescente diversidade de aplicações, à existência de diferenças drásticas no comportamento de utilizadores, à mobilidade de utilizadores e equipamentos, à complexidade dos mecanismos de geração e controlo de tráfego, e à crescente diversidade dos tipos de acesso e respectivas capacidades. Neste cenário é inevitável que a gestão da rede seja cada vez mais baseada em medições de tráfego em tempo real. Devido à elevada quantidade de informação que é necessário processar e armazenar, é também cada vez maior a necessidade das plataformas de medição de tráfego assumirem uma arquitectura distribuída, permitindo o armazenamento distribuído, replicação e pesquisa dos dados medidos de forma eficiente, possivelmente imitando o paradigma *Peer-to-Peer* (P2P).

Esta dissertação descreve a especificação, implementação e teste de um sistema de medição de tráfego com uma arquitectura distribuída do tipo P2P, que fornece aos gestores de rede uma ferramenta para configurar remotamente sistemas de monitorização instalados em diversos pontos da rede para a realização de medições de tráfego. O sistema pode também ser usado em redes orientadas à comunidade onde os utilizadores podem partilhar recursos das suas máquinas para permitir que outros realizem medições e partilhem os dados obtidos. O sistema é baseado numa rede de *overlay* com uma estrutura hierárquica organizada em áreas de medição. A rede de *overlay* é composta por dois tipos de nós, denominados de *probes* e *super-probes*, que realizam as medições e armazenam os resultados das mesmas. As *super-probes* têm ainda a função de garantir a ligação entre áreas de medição e gerir a troca de mensagens entre a rede e as *probes* a elas conectadas. A topologia da rede de *overlay* pode mudar dinamicamente, com a inserção de novos nós e a remoção de outros, e com a promoção de *probes* a *super-probes* e vice-versa, em resposta a alterações dos recursos disponíveis. Os nós armazenam dois tipos de resultados de medições: *Light Data Files* (LDFs) e *Heavy Data Files* (HDFs). Os LDFs guardam informação relativa ao atraso médio de ida-e-volta de cada super-probe para todos os elementos a ela ligados e são replicados em todas as *super-probes*, fornecendo uma visão simples mas facilmente acessível do estado da rede. Os HDFs guardam os resultados detalhados das medições efectuadas a nível do pacote ou do fluxo e podem ser replicados em alguns nós da rede. As réplicas são distribuídas pela rede tendo em consideração os recursos disponíveis nos nós, de forma a garantir resistência a falhas. Os utilizadores podem configurar medições e pesquisar os resultados através do elemento denominado de cliente. Foram realizados diversos testes de avaliação do sistema que demonstraram estar o mesmo a operar correctamente e de forma eficiente.

keywords

Traffic monitoring, peer-to-peer, distributed control, distributed storage, distributed access, probing.

abstract

The characteristics of Internet traffic are becoming more and more complex due to the large and growing diversity of applications, the drastic differences in user behaviours and the mobility of users and devices, the complexity of traffic generation and control mechanisms, and the increasing diversity of the link type and quality. In such an environment, it is inevitable that network management tasks will rely heavily on (real-time) traffic measurements. Due to the large amounts of data that need to be processed and stored, measurement platforms have to become more distributed, allowing for scattered storage, replication and efficient retrieval of measurement data, possibly mimicking the peer-to-peer (P2P) paradigm.

In this dissertation we describe the specification, development and evaluation of a distributed traffic measurement system with a P2P architecture, which provide network managers with a tool to remotely configure third-party monitoring modules installed at different points of the network in order to perform test measurements. The system can also be used as a large-scale measurement infrastructure in a community-oriented network where Internet users may share some processing power and storage space of their machines to allow other Internet users (e.g. researchers) to perform measurements, to retrieve and share the obtained results. The system is based on a hierarchical overlay network organized in measurement areas. The overlay network is formed by two types of nodes, called probes and super-probes, which perform the measurements and store the measurement results. Super-probes have the specific role of providing connection among measurement areas and manage the exchange of messages between the network and the probes connected to them. The topology of the overlay network can change dynamically, with nodes being inserted and removed on-the-fly, and probes being transformed in super-probes and vice-versa, in response to changes in the available resources. The nodes collect two types of measured data: Light Data Files (LDFs) and Heavy Data Files (HDFs). LDFs store the average round-trip time from each super-probe to every element it is connected to and are replicated in all super-probes, providing a coarse but widely available view of the network status. HDFs contain the results of detailed measurements carried out at the packet or flow level and can be replicated at some nodes of the overlay network. Replications are spread over the overlay network taking into account the resources available at nodes, so as to provide high resilience to failures. Users can configure traffic measurements and search the overlay network for measurement data through the so-called client element. The various tests carried out in the system have shown that it performs correctly and efficiently.

Table of Contents

Table of Figures

Table of Tables

Chapter 1. Introduction

The complexity of data networks is growing very fast due to the diversity of technologies, applications and user behaviors. In such environments, a complete and updated knowledge of the network status is of fundamental importance for network administrators. Traffic monitoring systems [NMT] provide administrators with a tool to detect and respond to network events or behaviors that can have a significant impact on the network performance.

Traffic monitoring systems can be classified as active or passive [Pasztor2001] [Corral2003] [Grossglauser2001]. Typically passive systems are based in a single client element that captures the packets being transmitted or received over the network to which the element is attached. They simply perform the analysis of the traffic that flows through the network, without changing it. Usually, they are used to identify the type of protocols involved and to measure one or more characteristics of the traffic that flows through the measurement point, like the average rate, the mean packet size or the duration of the TCP connections. These systems involve measurement intervals that can stretch from several milliseconds to weeks or even months, thus forcing the storage and processing of huge data quantities. Examples of currently available passive monitoring systems are *tcpdump* [TCPdump], NeTraMet [NeTraMet] and NetFlow [NetFlow]. Active systems are usually based in a requesting element and in a target (responding) element that answers to the requests. These systems insert a small amount of traffic directly into the network to infer performance statistics. Usually, they are intended to provide network performance statistics between two distinct measurement points, like for example mean packet delay and packet loss ratio. Those statistics can be one-way statistics, when they refer to a single direction of traffic flow, and round-trip statistics, when they refer to traffic that flows in both directions. In these systems measurement intervals are in the order of seconds or minutes and they produce small amount of data quantities because they only process the packets of the traffic they introduce in the network. Examples of currently available active monitoring systems are *ping* [RFC792], J-OWAMP [JOWAMP] and Internet2 OWAMP implementation [OWAMP].

Traditional traffic monitoring modules either rely on a single probe [TCPdump] [Ethereal] [NTOP] [MRTG] or in a centralized architecture where a set of probes are controlled by a single manager [RFC4656] [RFC3917] [RFC1757] [NMT]. The single probe system only monitors the traffic at one location and, therefore, is not flexible enough for medium and large size networks. The centralized architecture is able to provide an accurate view of the network status. However, it relies on a single manager, which makes it vulnerable to failures. Moreover, it stores measured data at a single collector, which may consume significant bandwidth when downloading data from probes.

Decentralized architectures, like the ones of P2P file sharing systems, can help overcoming some of the limitations of the centralized ones. In P2P systems there is no need of centralized servers which removes the single point of failure. In these systems all nodes may connect to each other and the resources (storage space, computing power, etc) are distributed among all connected nodes improving scalability. The nodes can provide services to other nodes of the network which can be remotely accessed through the P2P overlay network. Additionally, the stored information can be replicated in different nodes

guarantying availability. Some disadvantages of the peer-to-peer technology [Wilson2002] are the redundant structure; the nondeterministic service requests due to the distributed form of communications; and the availability of resources is not guaranteed since nodes can disconnect at any time.

Nowadays, decentralized systems based on P2P architectures are very popular, namely the file sharing applications such as Kazaa [Kazaa], eMule [eMule] and the ones based on Gnutella protocol [Gnutella] [BearShare] [Gnucleus] [LimeWire] [Shareaza]. Other popular peer-to-peer application areas include instant messaging and distributed computing. Entirely new application areas for peer-to-peer technology are still being discovered.

Traffic monitoring systems can profit from the advantages provided by models based in decentralized P2P architectures. This will be the basis of our proposal of a traffic monitoring system based in a P2P distributed architecture.

1.1 Proposed system and objectives

A traffic monitoring systems with a novel peer-to-peer (P2P) architecture was proposed within the research team where the author developed his MSc work [Salvador2005]. The main objective of this work is to complete the specification of this proposal, develop and test the proposed monitoring system. This work relies on previous experience of the research team in the development of traffic measurement tools (see, for example, http://www.av.it.pt/jowamp [JOWAMP]).

The name of the proposed system is Distributed Traffic Measurement System with a Peer-to-Peer Architecture – DTMS-P2P. This proposal is based on a hierarchical distributed P2P architecture similar to Gnutella 0.6 [Gnutella]. The adoption of a P2P architecture allows high tolerance to failures and distributed storage of measured data, and is suited for traffic monitoring in wide area network environments. Moreover, access and querying of measured data can be performed using traditional P2P file sharing schemes.

Gnutella is a decentralized file sharing network and it is a fully distributed alternative to semi-centralized systems such as FastTrack (KaZaA) and centralized systems such as Napster. Two versions of the Gnutella protocol are commonly used and specifications for both are publicly available at [Gnutella]. The original version of the protocol (version 0.4) was proposed in early 2000 [Nullsoft] [Gnutella04]. Later, in 2001, a new version was introduced and it was named Gnutella protocol version 0.6 [Gnutella06]. In the first version of the Gnutella protocol all peers had the same functionalities (client and servers) and were at the same level. In version 0.6 a hierarchical approach was introduced and instead of treating every peer as client and server, some peers were now treated as ultra-peers, routing search requests and responses for peers connected to them. This variation of the protocol introduced an improvement in scalability. For additional details of the two versions of the Gnutella protocol, the reader is referred to the Gnutella specifications [Gnutella]. According to Wikipedia [Wikipedia] in the last quarter of 2007 Gnutella was the most popular file sharing network in the Internet. We have selected the Gnutella protocol version 0.6 as the basis of our implementation due to several advantageous characteristics this protocol provides, namely, its hierarchical architecture which improves

scalability and reduces the load on the peers due to message routing, redundancy which allows a peer to be reached from different routes and protocol simplicity.

The proposed traffic monitoring system aims at providing an infrastructure that allows any monitoring module that can be executed through command line, such as *tcpdump* [TCPdump], *ping*, *traceroute*, J-OWAMP [JOWAMP], and others [NMT], to be remotely configured to execute (active or passive) traffic measurements and to allow the retrieval of the obtained results. The system implements an overlay network where the results of the configured test measurements can be shared among the elements of the system and can be retrieved by any user. To improve the system reliability, these results may be replicated at various locations. The proposed system can also work as an alarm system on which a node informs another node of a change in the network status, whenever it occurs. Such a system provides network administrators with a tool to easily monitor large networks.

Moreover, DTMS-P2P can also be used as a large-scale measurement infrastructure in a community-oriented network where Internet users may share some processing power and storage space of their machines to allow other Internet users (researchers) to perform measurements, to retrieve and share the obtained results. However, some issues must be taken in consideration when using DTMS-P2P as such an infrastructure: for example, the preservation of the privacy of ISPs and users, the legal and proprietary ownership concerns, the prevention of measurements that might cause harm to the network or to users (for example, high rate of traffic could generate significant costs for users who pay for bandwidth by usage), and the verification of the integrity of data collected from not trusted sources. In [Claffy2006] some of these issues are identified and some measures that should be taken in consideration in such infrastructures to guarantee the security of users and providers are described. Several community-oriented infrastructures are in common use among researchers. Some provide dedicated hardware (for example, Skitter [Claffy1999], Ark [Ark], PlanetLab [Peterson2002] [PlanetLab] and RON [Andersen2001]) and others implement an @home-style distributed measurement network (for example, NETI@home [Simpson2004] [NETI@home] and DIMES [Shavitt2005] [DIMES]). In the @home-style approach it is provided a downloadable tool to be installed at the Internet users' home machines which allow them to share their resources to other users. Such an infrastructure can also be implemented with the DTMS-P2P system, which does not require costly funding since in the @home-style approach the resources are shared among all users not requiring dedicated hardware.

1.2 Organization of the dissertation

This dissertation is organized as follows. The next chapter presents the state-of-art of monitoring systems, focusing the study in systems based in peer-to-peer architectures, and gives an overview about some community-oriented measurement networks. An overview of the proposed system is given at chapter 3. Chapter 4 describes the proposed DTMS-P2P protocol. The chapter 5 makes a comparison between the proposed system and some P2P file sharing applications. The description of our implementation of the proposed system is given in chapter 6. Chapter 7 presents some evaluation tests performed to validate the implemented system. A summary of conclusions and some suggestions for possible future research concludes the dissertation.

Chapter 2. State-of-art of Measurement Systems and Community-Oriented Networks

In this chapter we present the state-of-art of measurement systems based on P2P architectures. For a complete list of measurement systems see [NMT]. We will describe M-coop (section 2.1.1), pMeasure (section 2.1.2), DEMS (section 2.1.3), and DipZoom (section 2.1.4). In this chapter, we also present some community-oriented networks. We will describe PlanetLab (section 2.2.1), DIMES (section 2.2.2) and NETI@home (section 2.2.3). In section 2.3 we will draw some conclusions.

2.1 P2P based measurement systems

2.1.1 M-coop

Srinivasan and Zegura [Srinivasan2002] [Srinivasan2003] proposed a distributed peer-to-peer system that can be used to obtain network performance information. With the proposed system, the authors were particularly interested in real-time measurements (distance metrics such as latency, hop count, etc.). The proposed system was named M-coop (or Measurement Cooperative).

Architecturally the system is an overlay network of computers running the M-coop software. For scalability, each node on the network is assigned an “area of responsibility” (AOR), defining a set of IP addresses for which it can answer queries. Measurements are taken by the endpoints in two ways, actively, by sending periodic probe packets to each other, and passively, by monitoring the traffic that traverses the network where they are attached. The measurements data obtained are stored with meta-information such as time of measurement, accuracy, etc.

2.1.2 pMeasure

Liu, Boutaba and Hong [Liu2004] [Liu2005] propose the pMeasure tool which is built on top of the Pastry [Rowstron2001] peer-to-peer network, but the system can be build on top of any other P2P network.

The pMeasure architecture consists of a set of nodes each running pMeasure both as a server and as a client. When running as a server, a pMeasure node receives measurement tasks from other nodes and processes them if resources are available. A pMeasure node running as a client can submit measurement tasks and retrieve results from the system. A pMeasure node depends on its P2P substrate for submitting measurement tasks, receiving measurement results, and locating other nodes when needed. It is the system’s responsibility to automatically locate necessary and appropriate nodes for measurement tasks. To provide security in the pMeasure system all nodes must possess a public/private key used to encrypt and decrypt the messages exchanged between the nodes of the system.

The system can be used to perform both passive and active measurements. The tool comprises a passive monitoring module, to compute traffic throughput and port activity, and an active monitoring module to measures One-way Delay, Round-Trip Delay, Connectivity, and Trace Route. Each pMeasure node is equipped with a build-in passive

measurement task, which maintains statistics about the network traffic present on the host for every 15 min. In each 15 min statistics, a series of detailed counters are maintained, e.g. the number of packets and bytes from a port, the number of packets and bytes through a network interface card (NIC), etc. These statistics can be provided to the host users and can be delivered to other pMeasure nodes when requested.

2.1.3 DEMS

Finkenzeller, Kunzmann, Kirstädter, Schollmeier [Finkenzeller2006] proposed the Distributed End-System Monitoring Services (DEMS). The system is more focused on passive measurements. It captures the traffic flowing through the network to which the node is attached, analyzes it, and publishes the results via a Peer-to-Peer (P2P) framework. Locally observable data may be everything from packet statistics like throughput and jitter to detailed information extracted via deep-packet inspection. The level of inspection may vary: It reaches from simply parsing pairs of source and destination IP addresses for calculating the throughput up to protocol data at application level.

DEMS does not assume prior topology knowledge but collects this information on-demand. Based on the network topology information, loss and delay characteristics of single network elements can be determined by correlating subsequent local measurements with measurements from remote DEMS peers. To avoid overhead, the only active monitoring DEM uses are the well known route discovery methods like traceroute or the ICMP record route option. All other information is gained by passively monitoring packets.

2.1.4 DipZoom

Rabinovich, Triukose, Wen, and Wang [Rabinovich2006] [Wen2007] [DipZoom] proposed a system that explores a new approach to provide on-demand measurements. This approach, named DipZoom (for Deep Internet Performance Zoom), facilitates a peer-to-peer network of measurement providers and requesters, and it uses ideas from file-sharing P2P networks in its service discovery. This proposal is a system where anyone can offer measurements from their computers and other computing devices, and anyone can request measurements.

The DipZoom system consists of three main entities: the measurement providers who install DipZoom measurement software on measuring hosts (referred as measuring points, or MPs) and make these hosts available for measurements; the measurement requesters who request measurements from the measuring points; and the DipZoom Core, which matches measurement requesters with providers, processes payments, and enforces security and trust mechanisms. The DipZoom centralized core is very similar to Napster, thus it could become a potential bottleneck and a crucial point of failure. If the centralized core is not available, the entire system cannot be used.

In the proposed system, the measuring points (MPs) advertise to the core their capabilities (the platform, the offered measurements, the pricing, the maximum rate of measurements the MP is willing to perform, etc.), announce their coming on-line, and perform measurements requested by the core on behalf of the clients. The core maintains the database of the MPs and keeps track of those MPs that are currently on-line. The clients

query the database for the MPs that fit their requirements (based on such characteristics as the geographical location, the autonomous system to which the MPs belong, the MP's operating system, the MP's connection bandwidth), submit measurements requests from selected MPs, and download the results.

The initial proposed DipZoom system supported *wget*, *ping*, *traceroute*, and *nslookup* as part of the standard MP. This system required the implementation of a "measurement plug-in" mechanism that would allow it to incorporate arbitrary new measurement tools.

2.2 Community-oriented networks

2.2.1 PlanetLab

PlanetLab [Peterson2002] [PlanetLab] is a widely infrastructure of machines distributed all over the world designed and operated to serve as a test bed for overlay networks where researchers can experiment a variety of new network services, including distributed storage, network mapping, peer-to-peer systems, distributed hash tables, content distribution networks, routing and multicast overlays, QoS overlays, scalable object location, scalable event propagation, anomaly detection mechanisms, and network measurement tools.

PlanetLab allows the development and deployment of new network technologies in a controlled environment, incorporating realistic topologies and behavior, by providing dedicated hardware. Most of the machines are hosted by research institutions, although some are located in co-location and routing centers and are all connected to the Internet.

Many researchers of different academic institutions and industrial research labs have used PlanetLab to develop new technologies and there are many other active research projects running on PlanetLab.

2.2.2 DIMES

DIMES [Shavitt2005] [DIMES] is a scientific research project, aimed to study the structure, topology and evolution of the Internet with the help of a volunteer community. The main objective of this project is to map the Internet at several levels of granularity. To do so, they created a distributed active measurement infrastructure that applies the @home-style measurement approach. This infrastructure is formed by a collection of machines on which users have installed a provided downloadable tool (DIMES agent) from the DIMES project web site [DIMES].

The DIMES agent performs Internet measurements such as *traceroute* and *ping* at low rates. The agent is supposed to not send any information about its host's activity/personal data, but only the results of its own measurements. The results of measurement tests are collect in a central server and then processed and made available to the community.

DIMES is widely used among many Internet users who volunteer to participate in the project by installing the DIMES agent on their machines.

2.2.3 NETI@home

NETI@home [Simpson2004] [NETI@home] is a passive measurement infrastructure that also uses the @home-style approach. Its software can be freely downloaded and installed by volunteered users, to collect network performance and workload statistics from their machines. The basic idea is to sniff packets sent from and received by the host and infer performance metrics based on these observed packets. The software sends the resulting data to a server at the Georgia Institute of Technology (Georgia Tech), where they are aggregated to respect privacy and then made publicly available. NETI@home users are able to select a privacy level that will determine what types of data will be collected and reported. NETI@home is designed to run quietly in the background using few resources, with little or no intervention by the user.

2.3 Summary

This chapter describes existing measurement systems based on a P2P architecture. M-coop, pMeasure and DEMS are all equipped with built-in measurement modules used to perform active or passive measurements. Thus, a user is not able to install and use third party monitoring modules to perform measurements. On the contrary, DipZoom may support arbitrary monitoring modules, but requires the development of plug-ins to support them. Moreover, this system is based in a centralized core which is a critical point of failure.

All these systems have rudimentary storage capabilities, which make it difficult to handle large data files and restrict the type of measurements that can be carried out. The results of the test measurements are only stored at the nodes where the measurements were carried out, thus not guarantying availability and the possibility of multiple sources download. Moreover, the architectures are not hierarchical and, therefore, do not scale well with the number of measuring nodes.

Our proposal will try to overcome some of the limitations of the presented systems.

In this chapter, we also described some community-oriented networks. These networks are becoming very popular and they allow users to cooperatively perform measurements in the Internet and retrieve the obtained results. However, some issues must be taken in consideration when implementing such an infrastructure: for example, the preservation of the privacy of ISPs and users, the legal and proprietary ownership concerns, the prevention of measurements that might cause harm to the network or to users, and the verification of the integrity of data collected from not trusted sources. Additionally, in the @home-style measurement networks, researchers must ensure that the distribution sites are secure so that users do not download software that has been tampered with. In [Claffy2006] some of these issues are identified and some measures that should be taken in consideration in such infrastructures to guarantee the security of users and providers are described.

Due to its distributed P2P architecture the DTMS-P2P system can be used as a community-oriented measurement infrastructure where user allow other users to perform measurement tests in their machines and share the results among them.

Chapter 3. Overview of the DTMS-P2P Protocol

In this chapter we provide an overview of the various features of the DTMS-P2P protocol. The DPMS-P2P protocol creates an overlay network using TCP connections to allow the remote configuration of measurement sessions in the third-party monitoring modules installed at the overlay nodes, the replication of measurement files in some of these nodes, and the search and retrieval of the measurement results.

In section 3.1 we describe the way the overlay network is constructed and maintained. In section 3.2 we present the mechanism for configuring measurement sessions. In section 3.3 we identify the types of measured data files and we explain how the data files are replicated. In section 3.4 we describe the way the files can be searched and retrieved. Finally in section 3.5 we refer briefly to the security features of the protocol.

3.1 Overlay network

The overlay network consists of two main entities: the client and the node, which can be either in probe or super-probe mode. The client is the interface between the monitoring system and the user. It is used to configure the system, to configure the measurements and to retrieve the measured data. Clients do not require large processing or storage capabilities and, therefore, can run on simple devices such as PDAs or mobile phones. Nodes perform the measurements and store the results and, therefore, require more processing and storage capabilities than those required by clients.

To improve the scalability of the system, the nodes are organized in measurement groups, and each group is responsible for monitoring a particular network area. Measurement groups are identified by a unique id called measurement group ID or simply Group ID (appendix A.1). Each group has at least one super-probe, and can also have probes. Super-probes connect to each other and are responsible for communication with other measurement groups. Super-probes are identical to Gnutella's ultra-peers [Gnutella06]. A probe connects directly to a super-probe of its measurement group. Both probes and super-probes can perform measurements, but super-probes have also to manage the probes under its control and connect to other super-probes. Thus, super-probes must be the nodes with more resources (e.g. CPU usage, free memory and storage capacity) within a measurement group. A node can alternate dynamically between the two modes of operation, probe or super-probe, in order to adjust to different network conditions and resource availability. The node can be administratively configured to start operating as a probe or as a super-probe.

Figure 1 represents the hierarchical relationship between the system elements.

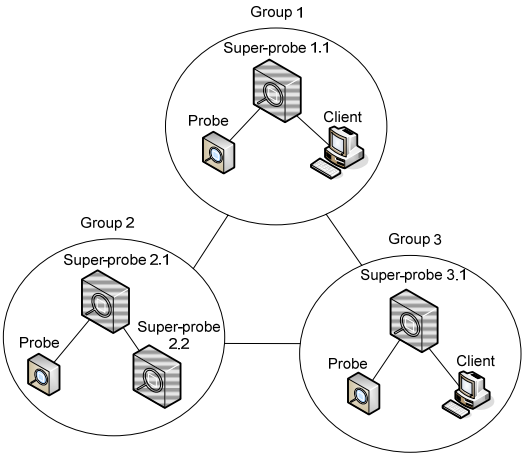

Figure 1. System hierarchy.

The overlay network is formed by connections between clients and super-probes, between probes and super-probes, and between super-probes (Figure 2). All other types of connections are not allowed, for example, connections between probes or connections between a probe and a client. Furthermore, clients and probes can be connected to only one super-probe which must be of the same measurement group. The measurement group of a node is defined by the node manager. The measurement group of a client is defined by the user.

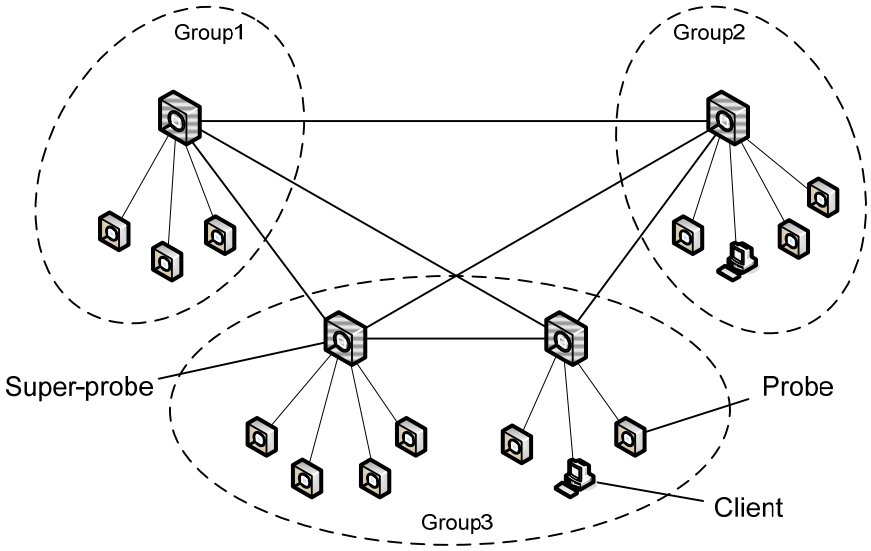

Figure 2. Overlay network.

Any node that wants to connect to the network must know at least one node that is already part of the network. If the node is the first one, it will promote itself to super-probe mode, if not yet in this mode, and will establish the first measurement group. It can then accept connections from clients, from probes of its measurement group or from other super-probes. When a node of a measurement group that still does not exist attempts connection to the network, it will promote itself to super-probe mode, if not yet in this mode. The network is dynamically built using this process.

All network elements maintain a cache of known nodes (CKN) including (hopefully) all the nodes of the network (probes and super-probes). For each node, the CKN includes information on its IP address, mode and measurement group. Moreover, each node in the CKN will have a status of “tested”, if the cache owner was able to test the connection with that node successfully, or “non-tested”, if a connection is yet to be attempted. Network elements try to connect to the closest nodes based on measured RTT.

The nodes are assumed to be highly heterogeneous in their processing and storage capabilities. Thus, it cannot be assumed that each node knows all other nodes in the overlay network, especially if the network is large. As a result connectivity between all network nodes is not fully guaranteed, but will be highly probable for reasonable sizes of the CKN.

The mode of a node (probe or super-probe) may change over time. Thus, in the process of connecting to the network, any element (client, probe or super-probe) will attempt connection to any node (probe or super-probe). Clients and probes may try connecting to probes of their measurement group and super-probes will try connecting to probes of its measurement group or others, even if these connections will not be completed because the remote nodes are indeed in the foreseen mode.

Clients and probes only test the nodes in their CKN when trying to connect to the network. This process ends upon discovery of the first super-probe of their measurement group that accepts the connection request. Thus, a client or probe that is connected to the network may have in its CKN both “tested” and “non-tested” nodes.

Super-probes must establish connections with all other super-probes in the network. Thus, super-probes must test all the nodes in their CKN when trying to connect to the network. However, a super-probe is considered to be connected to the network right after establishing the first connection to another super-probe of the overlay network.

The CKN needs to be updated frequently to reflect the most recent network topology. Towards this end, any element trying to connect to the network will exchange caches with the nodes it tries to connect to, even if the connection will not be completed. Moreover, super-probes will test the connections with nodes in its CKN periodically and, afterwards, will broadcast its cache to all network elements.

During the update process the cache can have some of its nodes removed, information regarding some of its nodes (mode, measurement group or status) updated or new nodes inserted. The update of a (local) CKN based on information received from a remote node must follow a number of rules to assure that the update reflects the most recent topology.

Super-probes will be demoted to probes whenever possible to keep the number of super-probes in each measurement group at a minimum. Two super-probes of the same measurement group connected to each other must determine if one of them can be demoted to the probe mode based on their available resources (memory occupancy and number of connections that can be accepted) and on the supported security modes. This verification is carried out in two situations: (i) periodically between each super-probe and all other super-probes of the same measurement group it is connected to and (ii) immediately after a successful connection between two super-probes of the same measurement group. The

demotion only occurs if one of the super-probes can accept the connection from all the elements currently connected to the other one. In case of demotion, the demoting super-probe must inform all the elements connected to it to connect to the surviving super-probe.

The load that each probe imposes on the super-probe it is connected to can vary over time. Super-probes can close connections to some of its probes in order to prevent becoming overloaded. The probes will then try to connect to other (less loaded) super-probes or, if not possible, promote themselves to the super-probe mode. This mechanism guarantees load distribution when a super-probe gets overloaded.

3.2 Measurement sessions

The overlay network provides the infrastructure that allows monitoring modules that can be executed through command line, such as *tcpdump* [TCPdump], *ping*, *traceroute*, J-OWAMP [JOWAMP], and others [NMT], to be remotely configured to execute (active or passive) traffic measurements.

In the initial configuration of a node, the node manager must indicate which monitoring modules the node supports and what the restrictions that must be respected are. One example of a restriction is to not support the option *-t* (Ping the specified host until stopped) when executing the *ping* command in windows machines. Another example is, in J-OWAMP, to use only the ports for the Server, Session-Sender and Session-Receiver that were previously configured by the node's manager.

The client is used as the normalized interface to configure the monitoring modules. Through the client, the user may introduce the address and measurement group of the node and the name of the monitoring module where he wants to perform the measurement. It can also introduce the configuration parameters of the measurement including, for example, the start time of the measurement. There are a number of features that can help users when they are unsure about one or more aspects of the measurements to be performed. For example, the client may obtain information on all nodes that support a given monitoring module or on all monitoring modules that a given node supports. The client may also obtain the usage description of a module. After the selection of the node and monitoring module, the client shows the user the list of restrictions that apply to that module on that node. This information can then be used in configuring the measurements.

3.3 Types of measured data files and data replication

There are two types of measured data files: light data and Heavy Data Files.

The Light Data File (LDF) provides a coarse, but updated and fully available, view of the network status. It stores the round-trip time (RTT) statistics between a super-probe and all the elements connected to it (clients, probes and super-probes). The LDF is periodically built by each super-probe and broadcasted to all other super-probes of the overlay network. Any super-probe will store only the most recent LDFs of all super-probes (including its own). Since clients download the LDFs using HTTP, they can be used to retrieve the LDFs stored at the super-probe they connect to or at any other super-probe.

The Heavy Data File (HDF) stores the results of all scheduled measurements, and can include detailed packet or flow information, and statistics of packet delays and losses measured over a period of time, obtained through active or passive monitoring techniques. Thus, for each configured measurement there will be one HDF storing the measurements. Moreover, HDFs may also store statistics, or even parameters of traffic models, calculated over the measured data, if the nodes are configured to do so.

These HDFs are stored at the node that created them and are possibly replicated at other nodes of the same measurement group (preferably) and/or other measurement groups (super-probes included). It is given preference for replication of files in the measurement group that created them since it is more likely that clients search measured data of the measurement group they are attached to. The replication improves the system reliability, since data can be retrieved even if the node that made the measurements becomes inactive or inaccessible. Also, a file can be more efficiently downloaded if simultaneously downloaded from multiple sources storing it, a process called multi-sources download.

By default a HDF should be replicated at least in one location. The maximum number of replications is configured by the node manager of the (original) source node, which is also given the option to define the maximum number of replications as a function of the file size.

The replication mechanism tries to achieve the following goals:

1. The replications must first be attempted in the measurement group of the (original) source node; only if there are not enough resources in this group should other measurement groups be searched for file replication.

2. The replicas must be (geographically) spread as much as possible among the nodes that have sufficient resources to replicate the file in order to guarantee availability.

In order to control the replication process we define time periods, which are relatively long (24 hours by default), and will be called replication periods. The (original) source node must try to create all replicas of the file during a replication period.

To process the required number of replications of a new HDF, the (original) source node must first search for possible locations where the file can be replicated by flooding a request to the overlay network. There are two possible scopes for the replication search process. The search can span over all super-probes (global scope) or only over the nodes of the (original) source node measurement group (local scope). In order to accomplish goal 1 stated above, the scope of the search process is made local in the first replication periods (in the first two by default) and only in the remaining ones (the third and the fourth by default) will be made global. In the search process, each super-probe that matches the replication search scope, must try to find among its probes a number of them that may store the file to be replicated and then respond to the request with the obtained list of probes. Super-probes must only return the number of locations solicited in the request message. To select the best locations and to speed-up the search process, super-probes can try to find twice as many the required number of locations among its probes and only return the ones with more available resources and, in case of a tie, which are faster. The

super-probe itself can be included in the returned list, when the super-probe has enough resources to replicate the file and there are not enough probes with available resources.

Whenever a response from a super-probe is received, and if the maximum number of replications has not been reached (for this purpose, the number of replications equals the number of replicas successfully created plus the number of replications being processed), the (original) source node selects a small number of nodes (2 by default) from the list sent in this message and asks these nodes to initiate the download of the HDF from the available storing nodes. The available storing nodes are the (original) source node and other nodes where the HDF was already successfully replicated. When there are two or more available storing nodes, the HDF can be downloaded simultaneously from those nodes (multi-source download). The (original) source node will store the addresses of the potential storing nodes listed in the response message for which a request has not yet been issued; this information may be used later to complete the required number of replications.

The storing node must send a confirmation to the (original) source node, when the download is finished or when an error occurred. This confirmation (positive or negative) will trigger new replication requests, again directed to a (small) number of nodes, if the maximum number of replications has not been reached and there are still potential storing nodes for which a replication request has not been issued. If there are not enough potential storing nodes, the (original) source must flood a new request to the overlay network asking for (new) potential storing nodes. A confirmation received from a storing node must trigger new replication requests because it is not possible to determine how many super-probes will respond to the potential storing nodes search request sent by the (original) source node and when responses will be received. The address of the storing node from where a positive confirmation was received is stored as a new storing node by the (original) source node and may be used as a new source from where the HDF can be downloaded in future replications.

The procedure of allowing only a small number of replications whenever a new request is issued to a super-probe, tries to avoid the concentration of many replicas in nodes under the control of a single super-probe, and spread the replicas in nodes that are geographically away from each other.

The procedure described above forces the start of new downloads to be dependent on the reception of response messages from super-probes or confirmation messages from nodes selected to download a file. Thus, if these messages stop arriving before the replication process is complete, the process will be halted. However, when a new replication period is started, the replication process can be resumed. At this time, we discard all requests for which a confirmation was not yet received. Afterwards, if there are still potential storing nodes for which a request was not issued, the (original) source asks a (small) number of these nodes to start the file download; otherwise it floods a new request to the overlay network for potential storing nodes. Thus the replication period serves two purposes: first it allows implementing a progressive search for potential storing nodes, starting by the measurement group of the (original) source node; second it provides a mechanism for resuming the replication process when it is halted due to unresponsive potential storing nodes or lack of potential storing nodes.

After the first replication period, there should be some more replication periods (one more by default) where the scope of the replication will still be local. This is because, there still may be potential storing nodes in the measurement group of the (original) source node after a replication period, since super-probes only include in its response to a request issued by the (original) source node a number of probes that is equal or lower than the maximum number of replications.

The replication period is defined as a long timeout because it is not possible to determine how long it will take for a file to be downloaded by a given selected node. This is because the duration of the download process depends on many factors (receiving and storing nodes speed and load, available network bandwidth, current network traffic, etc.). Still, it is very likely that a file will be completely downloaded much before the end of a long time period (e.g. 24 hours).

One alternative to having a global long replication period would be to have an individual long replication period for each file download. This would allow a better control of the number of replications to be performed. Recall that at the end of a global replication period new download requests will be issued, while ignoring on-going downloads, which may result in a number of replications higher than the maximum, if some of this on-going downloads get completed successfully. However, it would be much more complex to determine when to change from a local scope to a global scope replication.

The replication process stops when all replicas are successful produced or the total number of replication periods has been performed.

3.4 Search and retrieval of measurement files

The search and retrieval of measurement files can be performed through the client element. The methodologies adopted for these two functions are similar to the ones used in traditional P2P file sharing applications (e.g. Gnutella 0.6 [Gnutella]).

To initiate a search a client must first send a search request to its super-probe. The request includes the search criteria used to identify files the client is looking for. The search criteria may be the name of the file to be searched or a set of keywords. Any receiving super-probe should forward the request to all nodes connected to them. The nodes sharing files satisfying the search criteria should answer to the request. After receiving the responses to the configured request, the user can choose which files should be downloaded.

The data download is performed directly between the requesting element and the nodes where the files are stored using HTTP, out of the overlay network. This download can be made from multiple sources when the file is stored in multiple locations.

3.5 Authentication and/or Encryption

The exchange of messages in the overlay network and of data files can be authenticated and encrypted. There are three security modes: unauthenticated, authenticated and encrypted. In the unauthenticated mode both control messages and data files are sent as clear text; in the authenticated mode, data files are sent as clear text and control messages are both authenticated and encrypted; in the encrypted mode, both control messages and

data files are authenticated and encrypted. In order to implement the security modes, each connection between elements must share a secret key.

Chapter 4. DTMS-P2P Protocol Specification

In this chapter we present the detailed specification of the DTMS-P2P protocol. In section 4.1 we describe the supported security features. In section 4.2 we present the message flooding mechanism. In section 4.3 we describe the process of construction and maintenance of the overlay network. In section 4.4 we explain how monitoring modules can be remotely configured to process test measurements and how the measurement results are stored. In section 4.5 we describe how the data files with the measurement results are replicated. In section 4.6 we explain how these files can be searched. In section 4.7 we present the proposed download mechanism. In section 4.8 we describe the process to produce, broadcast and retrieve the information stored in the Light Data Files. In section 4.9 we present some additional functionalities implemented by the DTMS-P2P client. In section 4.10 we refer briefly to some considerations about additional security issues. Finally, in section 4.11 we will give a summary of the chapter and present the main conclusions. The complete description of each control message is provided in appendix A.2.

4.1 Security mechanisms

In the DTMS-P2P protocol the messages exchanged between network elements may be sent in their original format or may be authenticated and/or encrypted. The security features of the DTMS-P2P protocol are similar to those of the One-way Active Measurement Protocol (OWAMP) [RFC4656]. There are three security modes: unauthenticated, authenticated and encrypted. An element may support one or more security modes at the same time. It is up to the element's administrator to decide which security modes the element should support.

In the unauthenticated mode all messages are sent in their original format. In the authenticated and encrypted modes the control messages exchanged between elements of the system are both authenticated and encrypted. The data messages are not authenticated in both security modes but are encrypted in the encrypted mode.

The next subsections describe in more detail the authentication and encryption process and their relationship with the three different modes.

4.1.1 Message authentication and encryption

Message authentication is used to verify if the message was indeed sent by the remote element and if its content has not been modified. The DTMS-P2P protocol uses HMAC (keyed-Hash Message Authentication Code) for message authentication [RFC2104]. The message content (header and payload) together with a secret key (that we will designated by HMAC key) are applied to the hash function to produce a block of data that is inserted in the end of the message. In order to verify the authentication of a received message, an element must compare the HMAC block received in the message with the HMAC block it computes over the received message content. The two blocks must be equal otherwise the message authentication has been compromised. It is crucial to check for this before using the message; otherwise, existential forgery becomes possible. The complete message for which HMAC verification fails must be discarded and the connection where the message

was received must be dropped. In the DTMS-P2P protocol the hash function is SHA-1, the HMAC block is truncated to 16 bytes and the HMAC key has 32 bytes.

Message encryption is used to hide the message content. In the DTMS-P2P protocol messages are encrypted using the Advanced Encryption Standard algorithm in Cipher Block Chaining mode (AES-CBC). AES-CBC requires a secret key (that we will designate by AES key) and an initialization vector (IV) of the same size, which in the case of the DTMS-P2P protocol is 128 bits. In order to decrypt a received message, an element must know the AES key and the IV used by the sending element. The same key but different initialization vectors are used in each direction of the communication. To be able to use AES for stream encryption, all messages must have a size multiple of 16 octets. When required, a message must be padded to ensure this size, before authentication and encryption. The message padding must be removed in the receiving side, after the message decryption and message authentication.

In the DTMS-P2P protocol message encryption always requires message authentication, and authentication happens before encryption. Therefore, the HMAC block is encrypted along with the message content. This process is illustrated in Figure 3.

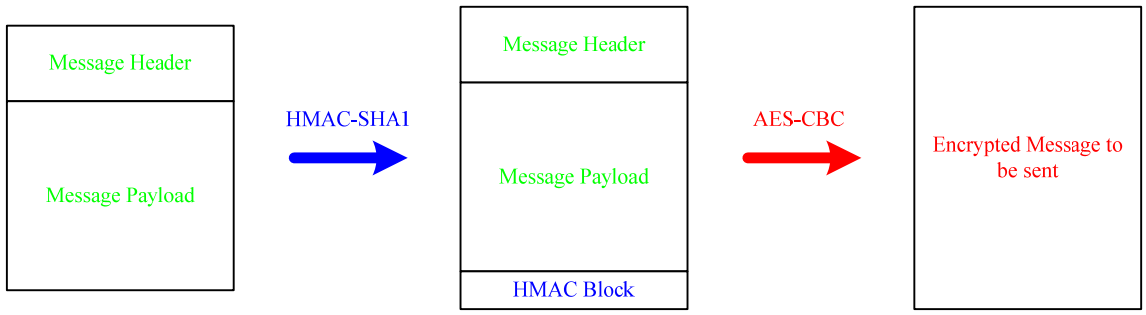

Figure 3. Message authentication and encryption.

In the authenticated and encrypted modes all control messages exchanged between peers are both authenticated and encrypted. The data messages are not authenticated in both security modes but are encrypted in the encrypted mode (section 4.7.4).

In the DTMS-P2P system each pair of elements uses different AES and HMAC keys and different IVs for communication, which are generated and exchanged when they connect to each other. Thus, an encrypted message that arrives at a node must first be decrypted, using the key of the connection with the previous element. Only then can the message be sent to the next element (possibly) encrypted using the key of the connection with that element.

4.1.2 Key generation and distribution

In the DTMS-P2P protocol the requesting element is responsible for generating the HMAC and AES keys used for authentication and encryption. These keys are then sent to the remote element as part of the connection set-up process, in a transmission encrypted with another key derived from a secret shared by both elements (that we will designate by session key). This encryption uses again AES in CBC mode but with an initialization vector of zero.

The secret shared by both elements is a pair of username/passphrase provided by the element's administrator. Both the username and passphrase must be a string of alpha numeric characters and must not contain new lines. Each element may be configured to support more than one username/passphrase pair. The different pairs of username/passphrase are provided by the element's administrator through an XML file with the Document Type Definition (DTD) represented in Figure 4. This file will be designated by File of Username/Passphrase Pairs (FUPP).

```
<!DOCTYPE PassPhrases [
  <!ELEMENT PassPhrases (PassPhrase*)>
  <!ELEMENT PassPhrase (userName, passPhrase)>
  <!ELEMENT userName (#PCDATA)>
  <!ELEMENT passPhrase (#PCDATA)>
  <!ATTLIST PassPhrase id ID #REQUIRED>
]>
```

Figure 4. File of Username/Passphrase Pairs XML DTD.

In this XML DTD the "PassPhrase" element represents a username/passphrase pair. It has two child elements: the "userName" element which contains the username of the pair and the "passPhrase" element which stores the correspondent passphrase. The "PassPhrase" element contains an attribute ("id") which stores the id of the correspondent username/passphrase pair.

Figure 5 shows an example of the File of Username/Passphrase Pairs. In this example there are two username/passphrase pairs. The first one has username "user1" and passphrase "pass1" and the second one has username "user2" and passphrase "pass2".

```
<?xml version="1.0"?>
<PassPhrases>

  <PassPhrase id="pp1">
    <userName>user1</userName>
    <passPhrase>pass1</passPhrase>
  </PassPhrase>

  <PassPhrase id="pp2">
    <userName>user2</userName>
    <passPhrase>pass2</passPhrase>
  </PassPhrase>

</PassPhrases>
```

Figure 5. Example of a File of Username/Passphrase Pairs.

The session key is derived from a passphrase using the PBKDF2 function [RFC2898]. In the case of the DTMS-P2P protocol, the PBKDF2 function applies HMAC-SHA1 to the input passphrase along with a salt value and iterates the process a number of times designated by count. The salt and the count are generated by the responding element.

4.2 Message flooding and routing

The DTMS-P2P protocol resort to flooding for routing messages from origin to one (or more) destinations. In the protocol there are two types of messages: request-response messages and information messages, which are sent one-way only (do not require a

response). Information and request messages are flooded towards one or more destinations while response messages are routed back to the source node using the reverse path of the request message.

The basic flooding rules are the following. Flooding occurs only at the super-probe level, except in the case of the super-probe directly connected to destinations. Messages are uniquely identified by a 16 byte random number, called Message ID, and a fragment index used when the content to be sent is fragmented in different messages (messages should not be larger than 4 KB – appendix A.2.1). When fragmentation is used all fragments have the same Message ID. Each super-probe maintains a table, called Route Table, storing for each message to be flooded it receives, the Message ID, the message's fragment index and the identifier of the connection where the message was received. When a message to be flooded arrives at a super-probe, the super-probe verifies if its Message ID and fragment index are in the Route Table. If yes, the message is discarded; if not the Message ID, the message's fragment index and the identifier of the connection where it was received are stored in the Route Table and the message is sent to all connections except the one where it was received.

The basic flooding process has one exception. When the message has a single destination, each super-probe that is visited by the message verifies if the destination node (super-probe or probe) is attached to it. If yes, it forwards the message to that node only.

The scope of flooding is constrained by a TTL (Time To Live) mechanism that prevents messages from being completely flooded in large networks. Specifically, each message has a TTL and a Hops fields (appendix A.2.1). At the source node the Hops is set to 0 and the TTL is set to the maximum number of super-probes the message is allowed to cross (usually 7). When a message is received by a super-probe the TTL is decremented and the Hops is incremented. If the TTL reaches 0, the message is no longer forwarded. Note that, in all nodes, TTL + Hops equals the initial TTL.

The routing of response messages back to the source node is performed as follows. Response messages have the same Message ID of the corresponding requests. When a response message arrives at a super-probe, its Message ID is searched in the Route Table. If it is found, the message is sent to the connection associated with the Message ID; otherwise it is discarded.

In the case of request-response interactions, to assure that the response message will reach the requesting element, it must have an initial TTL greater or equal to 1 plus the value of Hops received in the corresponding request message.

In the flooding process, when a message arrives at a super-probe and is to be forwarded to a probe directly connected to it, the super-probe first verifies if the message will be rejected by the probe. If yes, the message will not be forwarded to it. Super-probes maintain information about its probes that enables this type of verification. Consequently, whenever a probe receives from its super-probe a message not destined to it or a message with a request it does not support, the probe must reset the connection to its super-probe and try to connect to another super-probe of the network.

Whenever a message is received at an element, the element verifies its correctness. If the message is invalid (e.g. the message has an invalid field) the receiving element must discard the message and close the connection with sending element.

The size of the Route Table is a configurable parameter that is a tradeoff between memory occupancy and correctness of the flooding and reverse forwarding processes. Indeed, the record of a message (Message ID, message fragment index and connection identifier) must be kept in memory for time enough to assure that flooding is stopped or the response message in a request-response interaction is routed back to the requesting element. Clearly, when a new record arrives at a full Route Table, the new record must replace the oldest one.

Whenever a super-probe is the source of a message to be flooded, before sending the message to other super-probes, it must save a record of this message into its RouteTable. This should be done to prevent the super-probe from processing a message it sent.

A client must keep a list with the Message IDs of each message it sends (client's list of Messages - CLM). The CLM is used in three situations. First, it is used to verify, when a response message is received, if it is a response to a request made by the client. If not, the response message is discarded. Second, it is used to implement a wait mechanism: if the response is not received within a timeout period the corresponding record is removed from the list. Third, it is used when there is no timeout but the user may give up waiting for the response; in this case the client, when prompted by the user, removes the corresponding record from the list. The size of the CLM is a configurable parameter; when the CLM is full a new record must replace the oldest one.

The request-response message type of the DTMS-P2P protocol may be split in three different categories: (i) messages destined to one node of a given measurement group; (ii) messages destined to all nodes of a given measurement group and (iii) messages destined to all nodes of the network (all measurement groups). A more detailed description about how these messages are exchanged between the elements of the system will be presented in the next sections. At the end, we discuss the influence of the DTMS-P2P modes of security in the message flooding process.

4.2.1 Messages destined to one node of a given measurement group

When the message is destined to one node of a given measurement group (MG), the message must have information about the MG and IP address of the destination node.

Therefore, upon reception of a message a node (super-probe or probe) must first verify if the message is destined to it. The message's destination IP address must be used to process this verification. In case of success, the node must also verify if the message's destination MG is properly filed. In this case the message's destination MG must be equal to the node's MG. If not, the message must be discarded.

If the message is not destined to a receiving super-probe it must then verify if the message is destined to another node connected to it. The message's destination IP address must be used to process this verification. If the super-probe is directly connected to the destination node it must then verify the message's destination MG. However, this verification will

only be possible if the destination node belongs to the super-probe's MG. This is because super-probes do not save the information about the Group ID of super-probes of other MGs since, for the correct operation of the system, it is not imperative for a super-probe to know the Group ID of all the nodes connected to it. Saving these Group IDs would unnecessarily occupy the super-probe's memory space. Usually, super-probes only store the information about if the remote element belongs or not to its MG.

Thus, after an element receives a message, four situations may occur:

1. The message is destined to the receiving node's address, but not to its MG;
2. The message is destined to the receiving node's MG, but not destined to its address;
3. The message is destined to the receiving node's address and MG;
4. The message is not destined to the receiving node's address, neither to its MG.

In case 1, the receiving node must discard the message. In this case, if the receiving node is a probe or a super-probe of the sending element's (client or super-probe) MG, it must also reset the connection to the sending element. A client must not send erroneous messages. Super-probes must verify a message before forwarding it to a destination node. However, if the sending and the receiving nodes are both super-probes and of different MGs, the receiving super-probe must not reset the connection to the sending super-probe because, in this case, a super-probe is not able to verify if the message's destination MG is or not the MG of the destination super-probe.

In case 2, if the node is a probe it must discard the message and close the connection to its super-probe. If the receiving node is a super-probe four situations may occur: (i) the message is destined to one of the probes under its control; (ii) the message is destined to a super-probe of its MG and connected to it; (iii) the message is destined to a super-probe of another MG and connected to it and (iv) the message is destined to a node not connected to it.

In case (i) and (ii) the super-probe must forward the message directly to the receiving node only if it verifies that this node supports the request received in the message. If the destination node doesn't support the request, the super-probe must discard the message and may answer to the requesting element rejecting the request.

In case (iii), the message must be discarded. In this case the super-probe should not reset the TCP connection to the super-probe from where the message was received because a super-probe is not able to verify the message's destination MG if not directly connected to the destination node. However, if the message was received from a client connect to the receiving super-probe, the connection should be closed because the client is sending erroneous messages.

In case (iv), the super-probe must forward the message to all super-probes connected to it (except to the super-probe from where it received the message) and not only to the super-probes of the destination MG. This must be done because it may be possible that not all super-probes of the system are interconnected. This situation may happen because a super-

probe may be configured to only connect to a given maximum number of nodes of a DTMS-P2P network to prevent its overloading in very large networks, where the super-probe would have to process a lot of connections. Thus, forwarding the message to all super-probes connected to the receiving super-probe may guarantee that the message can reach the super-probes of the destination MG through another path, whenever necessary.

In case 3, the receiving node must process the message and, whenever required, may send a response message.

In case 4, if the node is a probe it must discard the message and close the connection to its super-probe. If the receiving node is a super-probe the same situations described for case 2 may happen. In case (i) and (ii) the super-probe must behave as in situation (iii) of case 2. In case (iii), the super-probe must forward the message directly to the receiving node. In case (iv), the super-probe must act as in situation (iv) of case 2.

The Figure 6 illustrates an example of the message flooding process when a message is destined to a given node of a given MG.

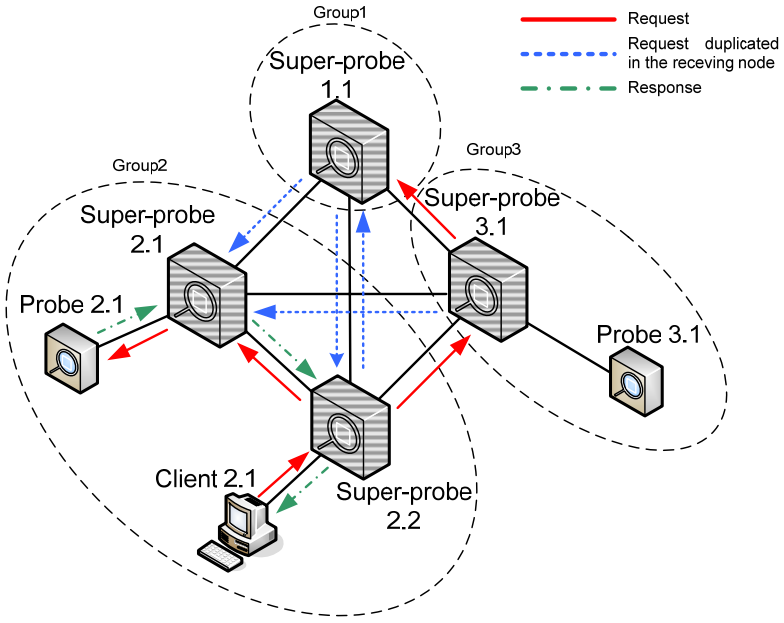

Figure 6. Message flooding process when a message is destined to a given node of a given MG.

In this example a request message is sent by the client 2.1 and is destined to the probe 2.1. The client 2.1 sends the request message to its super-probe. As the message is not destined to it and it is not directly connected to the destination node, the super-probe 2.2 should forward the received message to all super-probes connected to it. Thus, the message could reach its destination even if the super-probes 2.1 and 2.2 were not directly interconnected, but were indirectly interconnected through the super-probes of other MGs. Subsequently, each receiving super-probe should forward the message to all other super-probes connected to them, except to the one from where they received the request message. If a super-probe receives a request messages it received before, it should discard the received message. Because the super-probe 2.1 is directly connected to the node to which the message is destined, it should forward the message only to that node.

The probe 2.1 (destination node) upon reception of the request and whenever required, should process the request and send a response to the requesting element. The response message must have the same Message ID of the received request message. The response should be sent to its super-probe (node from which the probe received the request).

Using the information stored in the Route Table (Message ID of the request message and connection from where it was received), each receiving super-probe (super-probes 2.1 and 2.2) should forward the response only to the same node from where they received the request message. In this way, the responses follow back along the same path the request was received.

4.2.2 Messages destined to all nodes of a given measurement group

When a message is destined to all nodes of a given measurement group (MG), it must carry the Group ID of this MG. Each receiving node should only process the request received in the message if it belongs to the MG the message is destined.

In this case and for the same reasons described in situation (iv) of case 2 in section 4.2.1, a receiving super-probe must forward the message to all super-probes connected to it (except to the super-probe from where it received the message) and not only to the super-probes of the destination MG. Moreover, only the receiving super-probes of the destination MG and whenever required, should forward the message to the required probes connected to them and which will not discard the message.

The Figure 7 illustrates an example of the message flooding process when a message is destined to all nodes of a given MG.

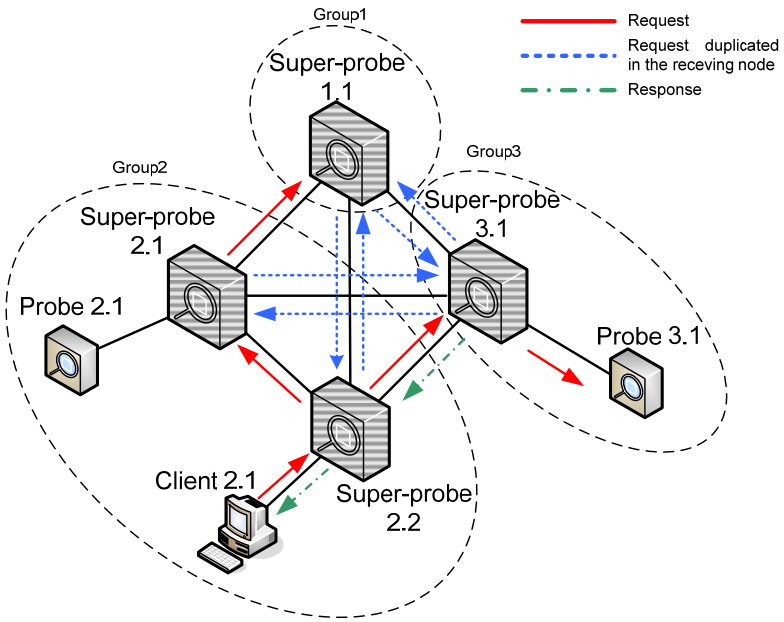

Figure 7. Message flooding process when a message is destined to all nodes of a given MG.

In this example a request message is sent by the client 2.1 and destined to the all nodes of the MG Group3. The client 2.1 sends the request message to its super-probe. After receiving it, the super-probe 2.2 should forward the received message to all super-probes

connected to it. Subsequently, each receiving super-probe should forward the message to all other super-probes connected to them, except to the one from where they received the request. Because the message is only designated to the nodes of the MG Group3, only the super-probe 3.1 should process the message and forward it to the probes connected to it (in this case, just the probe 3.1).

The nodes of the destination MG upon reception of the request and whenever required, should process it and send a response to the requesting element. In this particular case, is considered that only the super-probe 3.1 responds to the request. The response message must be routed back to the requesting element as described in the example of section 4.2.1.

4.2.3 Messages destined to all nodes of the network

When a message is destined to all nodes of the network (nodes of all measurement groups) all the receiving nodes should process the received request.

In this case, all receiving super-probes (independently of their measurement group) and whenever required, besides forwarding the received message to other super-probes, should also forward the message to the required probes connected to them that will not discard the message.

The Figure 8 illustrates an example of the message forwarding process when a message is destined to all nodes of the network (all measurement groups).

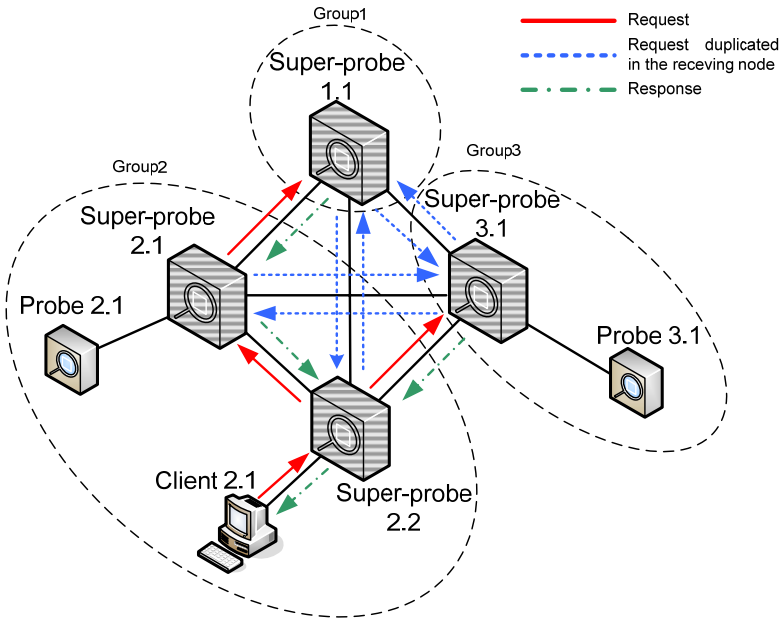

Figure 8. Message flooding process when a message is destined to all nodes of the network (all MGs).

In this example a request message is sent by the client 2.1 and is destined to all nodes of the network. The client 2.1 sends the request message to its super-probe. After receiving it, the super-probe 2.2 should process the message and forward it to all super-probes connected to it. Subsequently, each receiving super-probe should process the message and forward it to all other super-probes connected to them, except to the one from where they received the request. Whenever required, a receiving super-probe should also forward the

message to the probes connected to it. However, it may be possible that the request is only to be forwarded to super-probes or to both probes and super-probes, depending on the characteristics of the request message.

Each receiving node upon reception of the request and whenever required, should process it and send a response to the requesting element. The response message must be routed back to the requesting element as described in the example of section 4.2.1.

4.2.4 Message flooding vs. security modes

During the message flooding process after a request message is received an element should have in consideration the security modes supported by the requesting and destination elements before processing the received request and before forwarding the message to other node(s). Two situations may occur: (i) the request message can only be processed by the receiving node or forwarded to another element if the destination and requesting elements support a common security mode; (ii) the request message can be processed by the receiving node or forwarded to another element even if the destination and requesting elements do not support a common security mode. The case (i) occurs whenever the received request requires a direct connection between the two involved elements. Remember that two elements can only connect to each other if they support a common security mode. The case (ii) occurs whenever the received request does not require a direct connection between the two involved elements. This verification requires that messages include information about the security modes supported by their generating element (appendix A.2.1).

In case (i), if the receiving node is a probe it may not respond to the request and may close the connection to the sending super-probe. This is because in this case, a super-probe should only forward a message to a given probe, under its control, if it verifies that the probe will be able to process it, having in consideration the security modes supported by the probe and the requesting element. Super-probes must have information about the security modes supported by all nodes connected to them. This information is provided during the connection set-up process (section 4.3.5). However a super-probe must always forward a message to another super-probe even if the remote super-probe does not support a common security mode with the requesting element. This is because the message must reach all its possible destinations and this is only possible if super-probes perform this procedure, for the same reason explained in section 4.2.1 for situation (iv) of case 2.

The case (i) may occur in different situations. For example, only the nodes that support a monitoring module a user wants to use and support a common security mode with the requesting client, can be used by the user to perform a given test measurement. The elements should respect this procedure because the client will only be able to connect to the node and retrieve the results of a configured test session, if both elements support a common security mode. Another example may be found in the file replication process, where only nodes that support a common security mode with the (original) source node should be used to process a file replication. In this case, the (original) source node and the receiving node can only interconnect to replicate a file, if they support a common security mode. In these two examples, the receiving element must follow the procedure described in case (i) because the involved elements are supposed to interconnect to transfer a given

file between them. Files transference must always be processed outside the DTMS-P2P network directly between the two involved elements, using HTTP (section 4.7).

The case (ii) may also occur in different situations. For example, if a user wants to get the information about the available resources at a remote node, the node should answer to the request even if it does not support a common security mode with the client the user is using. In this case, the response is forwarded back to the requesting client using the DTMS-P2P network. The two involved elements are not required to directly connect to each other.

In the following figure is illustrated an example where different security modes (A – Authenticated, E – Encrypted and O – Open/Unauthenticated) are supported by the elements of the system and some elements support two different pairs of username/passphrase (1 and 2) shared secret. On each connection is depicted the security mode used between the two involved elements (for example, E(1) – encrypted security mode using shared secret 1).

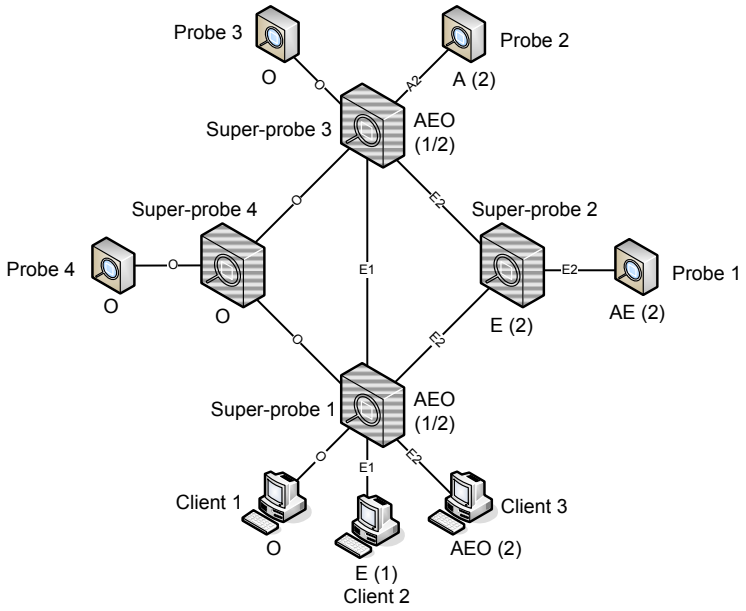

Figure 9. Message flooding vs. Authenticated and Encrypted security modes.

In this example, the client 1 should never be able to configure a test session at the super-probe 2 because it would not be able to retrieve the results from this node or from another node, if they are not replicated at a location supporting a common security mode with it. Also, the super-probe 2 should not consider the probe 2 as a possible location to replicate a file because it will not be able to interconnect with the probe to replicate the required file, since they do not support a common security mode. However, because of the reasons described for the case (ii), for example, if a user wants to get the information about the available resources at the probe 1, the node should answer to the request even if it does not support a common security mode with the client the user is using (for example client 1).

However, when the elements support different username/passphrase pairs, it may be possible that a client is able to configure a test session at a given node, but may not be able to retrieve the results from the selected node. This situation may happen because, although

the two elements support a common security mode, they do not support the same username/passphrase pairs. For example, if the client 2 is used to configure a test session at the probe 1, it may not be able to retrieve the results of the test session if the results are not replicated at a node supporting the same security mode and username/passphrase pairs the client 2 supports. To prevent this problem, the elements of the system should always know the same username/passphrase pairs.

4.3 Construction and maintenance of the overlay network

In this section we will describe the various mechanisms for the construction and maintenance of the overlay network. In section 4.3.1 we describe how a network is started. In section 4.3.2 we present the lists of known nodes and in section 4.3.3 the list of active connections maintained by each element. In sections 4.3.4 and 4.3.5 we describe how the elements connect to an already established network and what the protocol for connection set-up between any two peer elements is. In section 4.3.6 we present the topology maintenance mechanisms. Finally, in section 4.3.7 we describe the rules for updating the cache of known nodes.

4.3.1 Initial topology

To initiate a DPMS-P2P network, the network administrator must first start a node in super-probe mode. If the node is started in probe mode it will promote itself to super-probe mode; this mechanism will be described in section 4.3.4. The started node will create the first measurement group of the network. Subsequent network elements should be configured to connect to nodes already connected to the network. When a probe or client wants to connect to the network, they must connect to a known super-probe of their measurement group. A super-probe must connect to all other super-probes of the network, from the same or other measurement groups. If a started node belongs to a new measurement group, not yet present in the overlay network, it will be started as a super-probe (from this new group). In the DTMS-P2P network, both the probe and super-probe nodes must be always waiting for incoming connection requests. The client only requests connections. It should not receive connections requests from elements trying to connect to the network.

4.3.2 Lists of known nodes

Each network element maintains two lists with information about the nodes in the network, one in memory and another in disk. We will denote the list maintained in disk as the File of Known Nodes (FKN) and the list maintained in memory as the Cache of Known Nodes (CKN). In both lists, each entry refers to a node in the network that the list owner knows about and contains several fields. In the FKN there are four fields in each entry: measurement group, mode (probe or super-probe), IP address (DTMS-P2P supports both IPv4 and IPv6 addresses) and listening port. In the CKN there is an additional field in each entry: the average RTT (between the list owner and the node).

The CKN is structured in several parts according to three attributes: measurement group, mode (probe or super-probe) and status (tested or not tested). This is represented in Figure 10. First, the CKN is divided in two parts, one gathering nodes of the list owner measurement group and another gathering nodes of other measurement groups. We will

refer to the first part as the ownerMG and the second one as the otherMGs. The second part is further divided in several parts, each gathering nodes of a specific measurement group. Each of these parts is further divided according to status and mode. Each part corresponding to a measurement group will include first “tested” super-probes, second “tested” probes, third “non-tested” super-probes and last “non-tested” probes. Within each of the smallest parts the nodes are sorted according to RTT. Note that an element may be unable to compute the RTT to a node that is switched off, behind a firewall or configured not to answer to ICMP Echo Requests. Thus, nodes for which it was not possible to measure the RTT are placed in the end. When it is determined that a node must be removed from a part of the CKN, the last node of this part must be removed. The parts corresponding to each measurement group are sorted placing first the ownerMG and then the parts corresponding to other measurement groups sorted according to average RTT, where the average is computed over all group elements.

Using these rules, the CKN is sorted such that closer elements and closer measurement groups are placed first. This will enhance the probability that connected elements will be geographically near to each other and, in this way, improve the latency of the network.

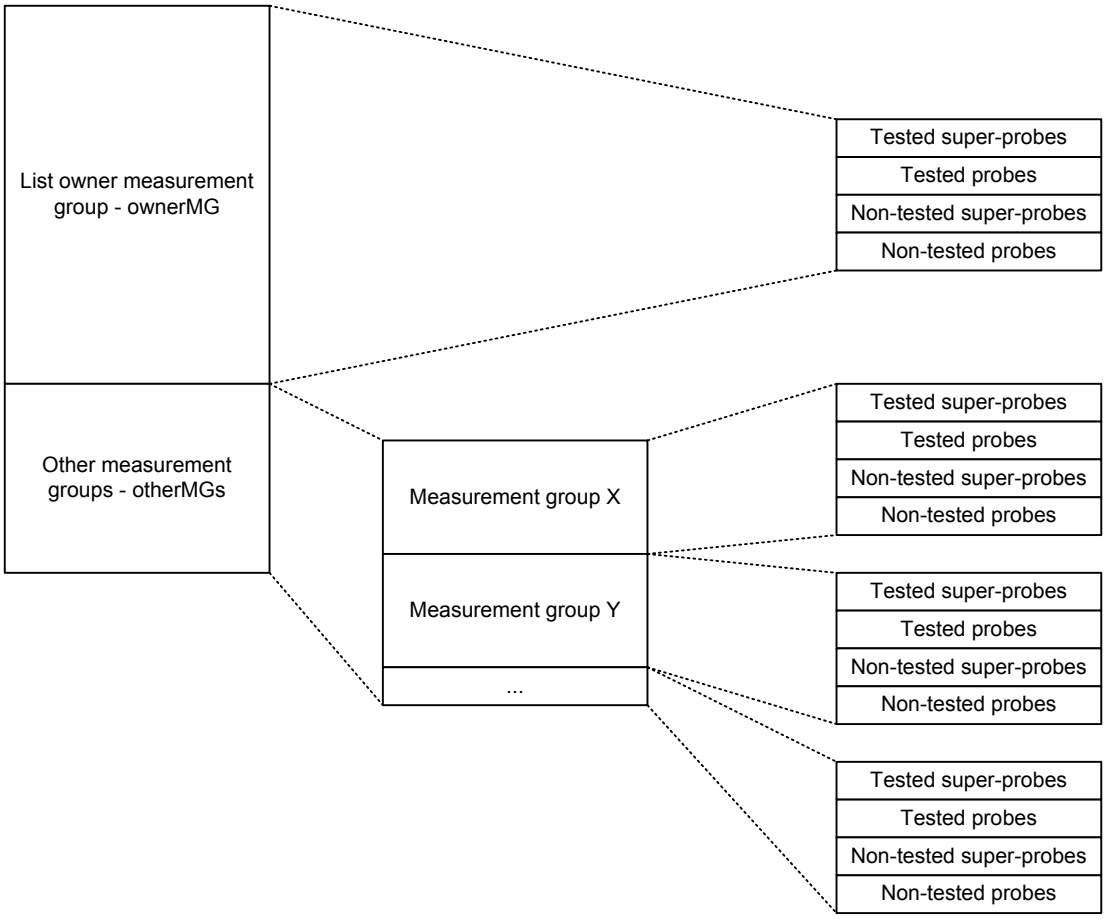

Figure 10. Structure of the CKN.

The CKN has a (configurable) maximum number of entries (maxNumberOfNodeAddr). Ideally the CKN should include all nodes in the network. However, this may not be possible for large networks. In this case, it is important that the CKN contains not only

nodes from the measurement group of the list owner but also nodes from every other measurement groups with as many groups as possible represented in the list. In this way we minimize the probability of having groups of nodes isolated from each other. The CKN is managed using a complete sharing policy. Under this policy, it may be possible that the list is completely filled with nodes either from the list owner measurement group or with nodes from other measurement groups. However, when the list is full, a (configurable) minimum of entries is assured for each part. This minimum is minOtherMGs for nodes from other measurement groups and minOwnerMG for nodes of the list owner measurement group. Note that these minima are correlated such that minOwnerMG = maxNumberOfNodeAddr - minOtherMGs. Moreover, in the case of nodes from other measurement groups there are a (configurable) maximum number of entries per group (maxOtherMGs).

The CKN structure favors assuring connectivity among all nodes. However, in large networks, it is neither required nor efficient to have full connectivity among super-probes. The limits described above help managing the connectivity of the overlay network.

When a network element is installed it should be provided with a FKN with at least one entry. The FKN will then be updated dynamically using information received from the network nodes that the list owner connects to.

We have adopted XML as the FKN format because of its readability which facilitates the exchange of files between different implementations of the DTMS-P2P protocol. The Document Type Definition (DTD) of the FKN is represented in Figure 11.

```
<!DOCTYPE FileOfKnownNodes [
  <!ELEMENT FileOfKnownNodes (GroupID*)>
  <!ELEMENT GroupID (super-probes,probes)>
  <!ELEMENT super-probes (node*)>
  <!ELEMENT probes (node*)>
  <!ELEMENT node (IPVN,IP,port)>
  <!ELEMENT IPVN (#PCDATA)>
  <!ELEMENT IP (#PCDATA)>
  <!ELEMENT port (#PCDATA)>
  <!ATTLIST GroupID id ID #REQUIRED>
]>
```

Figure 11. File of Known Nodes XML DTD.

In this XML DTD the “GroupID” element comprises the information about the nodes of a given measurement group. It contains two child elements: the “super-probes” element and the “probes” element. Both elements comprise the “node” element which stores the information about the IP address (“IP” element) and its version number (“IPVN” element) and the port number (“port” element) where a given node is running. The “GroupID” element has one attribute (“id”) which represents the GroupID of the correspondent measurement group.

Figure 12 shows an example of a FKN. In this example, the list has two measurement groups, one with Group ID 0 and another with Group ID 1 (appendix A.1). The first measurement group has one super-probe and two probes. Their IP version number, IP address and listening port number are given in the IPVN, IP and port fields respectively. The second measurement group only has one address corresponding to a super-probe.

```
<?xml version="1.0"?>
<FileOfKnownNodes>

  <GroupID id="00000000000000000000000000000000">
    <super-probes>
      <node>
        <IPVN>4</IPVN>
        <IP>192.168.0.1</IP>
        <port>22368</port>
      </node>
    </super-probes>
    <probes>
      <node>
        <IPVN>4</IPVN>
        <IP>192.168.0.2</IP>
        <port>22368</port>
      </node>
      <node>
        <IPVN>4</IPVN>
        <IP>192.168.0.3</IP>
        <port>22368</port>
      </node>
    </probes>
  </GroupID>

  <GroupID id="00000000000000000000000000000001">
    <super-probes>
      <node>
        <IPVN>4</IPVN>
        <IP>192.168.1.1</IP>
        <port>22368</port>
      </node>
    </super-probes>
    <probes>
    </probes>
  </GroupID>

</FileOfKnownNodes>
```

Figure 12. Example of a File of Known Nodes.

4.3.3 List of active connections

An element that has joined the network maintains a List of Active Connections (LAC). This list is used to store information related with the connections that the element maintains with other network elements. For each connection the following information is stored in the LAC: connection socket, time of socket creation, and IP address, port number, mode of operation and supported security modes of remote element.

Only the information related to active connections should be kept in the LAC. Whenever a connection to a remote element is closed, it must be removed from the LAC. However, if an element loses connection to a remote super-probe due to an unknown reason it should try to connect to it again shortly to verify if the remote super-probe is still active.

Clients and probes connected to the network can only have one active connection, to a super-probe of their measurement group. If these elements are not connected to the network the LAC must be empty. Super-probes can have connections to clients, probes and other super-probes of their measurement group and connections to super-probes of other measurement groups. There is a (configurable) maximum number of connections that a super-probe can have active, called maxNumberOfConnections. This comprises

connections to clients, to probes or to other super-probes of its measurement group or others. In addition, super-probes are only allowed a (configurable) maximum number of connections to probes, called maxNumberOfProbeConnections. The limit on the number of connections to probes a super-probe can accept is defined to reserve some space for connections to super-probes and clients. Note that the number of probes is likely to be much larger than the number of super-probes and the number of clients.

4.3.4 Network connection process

When an element is started it will try to connect to the network immediately.

A node can be in super-probe or probe mode if it was in that mode before a system failure or when set administratively. The measurement group of a node (probe or super-probe) is set by the node administrator. The measurement group of the client is set by the user. Thus, the user must select the measurement group close to where the measurements are to be performed or the measurement results are to be retrieved and must provide one or more nodes of this measurement group in the FKN.

When an element is started it first populates its CKN based on the FKN. If the element's address is in the FKN it must not add it to the CKN. Moreover, a node must not be duplicated in the FKN. For each entry in the FKN, the element measures the RTT to the corresponding node. The RTT is measured using the *ping* command [RFC792]. Note that an element may be unable to compute the RTT to a node that is switched off, behind a firewall or configured not to answer to ICMP Echo Requests. These nodes should be placed in the end of the CKN and will be the last ones to be tested. When a node is inserted in the CKN it will always be in the right order, which is described in section 4.3.7. If there is space available, the entry is simply inserted in the CKN. When the CKN becomes full, the entry may or may not be inserted, depending on the number of addresses in each part of the CKN and on the measured RTT. Recall that the CKN is split in several parts one for each measurement group. We will denote the part of the list that gathers nodes from the measurement group of the list owner by ownerMG and the set of parts of other measurement groups by otherMGs. When the entry is from a node of the list owner measurement group, then it will be inserted in the CKN if the number of addresses in part ownerMG is bellow its threshold or if its above the threshold and the measured RTT is smaller than the largest one in this part. In the first case, the last entry of the part of other measurement groups with the largest number of addresses will be removed. In the second case, the last entry of part ownerMG will be removed (first the element should try to remove probes, then super-probes). When the entry is from a node that is not from the measurement group of the list owner, then it will be inserted in the CKN if the number of addresses in part otherMGs is bellow threshold or if it is above threshold and the measured RTT is smaller than the largest one in the part of its measurement group. In the first case, the last entry of part ownerMG will be removed. In the second case, the last entry of the part of its measurement group will be removed. This process ends by dividing each part of the list in two additional parts, the first one gathering super-probes and the second one gathering probes.

The process of establishing initial connections varies according to the type of network element. Clients and probes must try to establish one connection with a super-probe of its measurement group. Moreover, a client only requests connections. Super-probes try to

establish connections with all other super-probes. In this process, a part of the list is always scanned sequentially, from the first entry (smaller RTT) to the last one. Scanning the CKN by increasing order of RTT will enhance the probability that the elements will be geographically near to each other and, in this way, improve the latency of the network.

The entries in the CKN are given an attribute of “tested” or “non-tested”, depending on whether a connection to them was or not tested by the CKN owner. Only “non-tested” nodes must be tested when connecting to the network.

The connection set-up process between two elements is described in section 4.3.5.

Clients or probes should first try to connect to a super-probe of its measurement group. If not successful, then they should try to connect to a probe of its measurement group. This should be done because the CKN may not reflect the most recent mode of a node, since nodes can switch dynamically between the probe and super-probe modes. Thus one or more nodes declared in the CKN as being in the probe mode may (already) be in super-probe mode. Note that a client or a probe should never try to connect to nodes of other measurement groups. Clients or probes are considered to have joined the network when they successfully connect to a super-probe of its measurement group. If this was not possible for a probe, it promotes itself to super-probe mode and reinitiates the process of connecting to the network (now as super-probe). If it was not possible for a client, it must stop operating.

A super-probe must try to connect to all super-probes of the network. It starts by testing the super-probes of its measurement group and then proceeds with the probes of its measurement group (for the same reason explained above). After testing all super-probes and probes of its measurement group listed in the CKN, the super-probe should also try the nodes (super-probes and probes) of other measurement groups. A super-probe stops the network connection process when all nodes in the CKN have been tested or when the maximum number of active connections the super-probe can maintain has been reached.

When an element is in the process of connecting to the network, new nodes or more recent information related with nodes already in the CKN may be discovered because, when connecting to a remote element, the element always retrieves its CKN. Thus, the element will update the CKN according to the criteria explained in 4.3.7. It will then proceed the scanning of the CKN and try to connect with the next node to be tested in the list. There is some exceptions to this behavior: (i) when an element is testing probes of its measurement group and new super-probes of its measurement group are discovered, the new super-probes must be tested before proceeding with the remaining probes; (ii) when an element is testing nodes of other measurement groups and new super-probes or probes of its measurement group are discovered, the new super-probes or probes must be tested before proceeding with the remaining nodes of the other measurement group and (iii) when an element is testing probes of a given other measurement group and new super-probes of this other measurement group are discovered, the new super-probes must be tested before proceeding with the remaining probes.

When a super-probe is in the process of connecting to the network, before trying to establish a connection to a remote node identified in the CKN, it must first verify if there is not a connection to that node in its list of active connections. This situation can happen if

the remote node was the first one to request the connection. In this case, the connection status of this node in the list will be changed to "tested" and the super-probe will proceed to the next node in the list. Note that this situation will not happen in the case of probes, because probes do not accept connection attempts from other nodes.

4.3.5 Connection set-up

Peer network elements connect to each other using a connection set-up process that will be described in this section.

The connection set-up process has three phases and is illustrated in Figure 13. In the first phase the requesting element establishes a TCP connection with the responding element. In the second phase the network elements negotiate the security mode and parameters that will be used in subsequent messages. In the third phase, the peer elements exchange information regarding measurement group, mode, IP address, listening port, and (local) CKN.

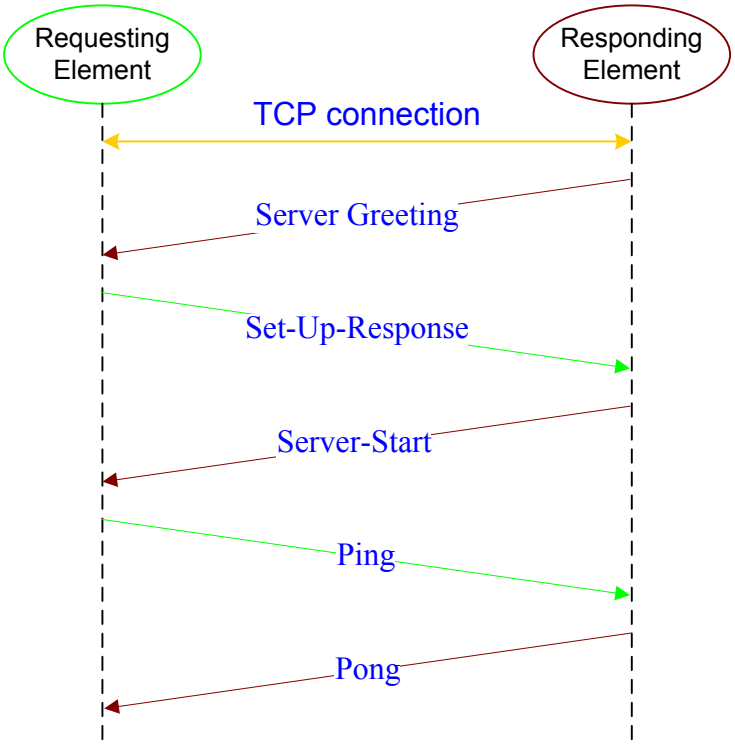

Figure 13. Messages exchanged during connection set-up.

4.3.5.1 Second phase

In the second phase, the responding element sends a Server Greeting message (appendix A.2.2), indicating its willingness to accept or not the connection request and the security modes it supports. If the connection request is not accepted the responding element closes the TCP connection after sending the Server Greeting message. The Server Greeting message also includes the salt and the count needed to compute the session key (section 4.1). The Server Greeting message includes a challenge used to authenticate the requesting element. This message is completely transmitted as clear text.

After receiving a Server Greeting message the requesting element responds with a Setup Response message (appendix A.2.3). First it must verify if it can support any of the security modes of the responding element. The strongest security mode must be selected. The encrypted mode is the strongest mode and the unauthenticated mode is the weakest one. If there is no agreement on the security mode, the requesting element may close the TCP connection and does not send the Setup Response message. If the authenticated or encrypted modes are selected the requesting element computes the AES and HMAC keys (section 4.1). These keys, together with the challenge received in the Server Greeting message, are sent encrypted with the session key in the Setup Response message. The message also includes the username of the username/passphrase pair used to compute the session key and the IV that will be used by the requesting element, sent as clear text.

The second phase of connection set-up finishes when the responding element sends a Server-Start message (appendix A.2.4). Communication may proceed if the requesting element is authenticated. Otherwise, the responding element closes the TCP connection after sending the Server-Start message. This message also includes the IV that will be used by the responding element, sent as clear text.

Since the requesting element may have more than one username/passphrase pair, in case of an authentication failure and if there is a username/passphrase pair in its FUPP that has not been tried, the requesting element reinitiates the connection set-up process with the same responding element using the next username/passphrase pair in the FUPP.

4.3.5.2 Third phase (handshaking)

In the third phase (handshaking) the peer elements exchange two control messages, called Ping and Pong.

Following the completion of the second phase the requesting element must send a Ping message (appendix A.2.5) to the responding element. This message is used to inform the responding element about the mode, measurement group, IP address and listening port of the requesting element. It also provides the responding element with the CKN of the requesting element. After receiving the Ping message, the responding element must update its CKN, as described in section 4.3.7.

After sending the Ping message, the requesting element must wait for a response, and if no response is received within a predefined timeout it must retransmit the Ping message a (configurable) number of times, called numberOfRetransmissions. When the maximum number of attempts is reached, the requesting element closes the TCP connection.

The response to a Ping message is a Pong message (appendix A.2.6). The Pong message is used by the responding element to notify the requesting element whether or not it accepts the connection and to provide the requesting element with information about its mode, measurement group, IP address, listening port and CKN. The CKN sent in the Pong message is the one immediately before the reception of the Ping message; it is not updated according to the CKN received in the Ping message, because it is useless to transmit back to the requesting element information that it already contains. After receiving the Pong message, the requesting element must update its CKN, as described in section 4.3.7.

The responding element will not accept the connection, and will notify the requesting element of the cause through the Pong message, in the following cases: (i) protocol failure; (ii) the responding element is already connected to the requesting element; (iii) the responding element is a probe; (iv) the responding element is a super-probe but the requesting element is a probe or a client from another measurement group; (v) the responding element is a super-probe and its maximum number of active connections has been reached; (vi) the responding element is a super-probe and the requesting element is a probe from its measurement group, but the former does not have enough resources (CPU and memory) or its maximum number of active connections to probes has been reached. In cases (i) and (ii) the Pong message must not include the CKN of the responding node. A protocol failure in the responding element will occur if the addresses of the requesting or responding elements are present in the field that transports the CKN. Recall that both Ping and Pong messages have separate fields for transporting the address and CKN of the element that sent the message. A protocol failure may also occur if one or more fields have invalid values. In case of rejection, the responding element must close the TCP connection with the requesting element. In case (ii), if the responding element is a super-probe and the requesting element is a probe or client, then the previous connection must also be closed. This is because a probe or client having this behavior would be violating the protocol, since the protocol only allows probes and clients to maintain one connection with a super-probe (of its measurement group). Otherwise, if the requesting element is a super-probe (irrespective of responding element) the previous connection is maintained since super-probes may accept and request connections to remote elements.

If the connection is accepted, it will be added to the LAC of the responding element.

The requesting element will discard the connection in the following cases: (i) the responding element rejects the connection; (ii) a protocol failure occurs; (iii) the requesting element is already connected to the responding element and (iv) the requesting element is now a probe and it is already connected to a remote super-probe. A protocol failure will occur in the same cases of the responding element; in addition it will occur if the Message ID of the received Pong message is different from the one of the transmitted Ping. Case (iii) may occur when the requesting element is a super-probe, since super-probes may accept and request connections to remote elements. Case (iv) may occur if the requesting element was initially a super-probe that, while processing the connection to the responding element, was demoted to a probe in result of a connection to another super-probe. In case of rejection the requesting element must close the TCP connection to the responding element.

If the connection is accepted, it will be added to the LAC of the requesting element.

When both nodes accept the connection the connection set-up process ends.

A super-probe involved in the handshaking process, either requesting or responding element, will always store information on the security modes supported by the remote element. This will be used in the configuration of measurement tests (section 4.4), in the file replication process (section 4.5) and in the results search process (section 4.6).

Since a network element may be processing several connection attempts at the same time, there is the possibility that two (or more) connections are added simultaneously to the

LAC, making the maximum number of active connections to be exceeded. In order to avoid this situation, the LAC only allows one access at a time (when a connection attempt is accessing the LAC the access of other connection attempts must be blocked) and access requests are queued and served using a FIFO discipline.

Note that the acceptance of a connection is independent of the CKN update process. A connection may be accepted even if it is not possible to store the address of the remote element in the local CKN.

While performing the connection setup process it may happen that a super-probe (*sp1*) requests a connection to another super-probe (*sp2*) exactly at the same time the remote super-probe (*sp2*) is requesting a connection to it. In this case, if both super-probes accept the connection request received from the remote one they will not be able to complete their own connection request since they will not be able to add it to their LAC after receiving the response from the remote one. This is because they will first add to their LAC the connection request received from the other one before sending their answer to the received request and only after receiving the response from the remote one will try to add the connection they requested to their LAC. But, the LAC can not have two connections to the same element. Thus, both connection requests will be closed by the requesting super-probe which is expecting that the connection requested by the remote super-probe will remain active, which will not be the case. Since an element must retry to connect to a remote super-probe whenever the connection to it is closed by an unknown reason (section 4.3.3) *sp1* will retry to connect to *sp2* after this last one closes the connection it requested to it. The super-probe *sp2* will have the same behavior. Thus, both super-probes will retry to connect to each other and now, hopefully, at different instants.

4.3.6 Topology maintenance mechanisms

In this section we describe the three mechanisms used to maintain the topology of the overlay network. The first mechanism tries to assure that each network element has always an updated view of the set of nodes that are active in the overlay network. The second mechanism tries to assure that the number of super-probes in the overlay network is kept at a minimum. The third mechanism tries to assure that the load of super-probes is distributed such that none gets overloaded. These mechanisms will be described in the following sections.

4.3.6.1 Maintaining the list of known nodes

In order to assure that network elements maintain an updated view of the set of nodes that are active in the overlay network the following mechanisms are used:

i. The elements should exchange the information stored in their local CKN when they connect to each other;

ii. Periodically, a super-probe sends its CKN to all nodes of the overlay network, using a Pong message (Pong flooding process);

The mechanism of case (i) was described in the connection set-up process (section 4.3.5.2). The other mechanism will be described in the next section.

4.3.6.1.1 Pong flooding process

In the Pong flooding process super-probes broadcast their local CKN to all other elements connected to the network by sending a Pong message (appendix A.2.6). This process is performed in three different situations: (i) when the super-probe finishes its network connection process (section 4.3.4), (ii) periodically (short interval, 2 minutes default), preceded by a test to all "non-tested" addresses in the super-probe's CKN and (iii) periodically (large interval, 30 minutes default), preceded by a test to all addresses of both the CKN and the FKN to which the super-probe is not connected to.

Pong flooding is required, in the first place, to assure that the arrival of new nodes to the overlay network is made known to all other elements. Moreover, the flooding of Pong messages is required to have some periodicity, because, due the (assumed) heterogeneity of network elements, namely in terms of storage capacity, it is not possible to guarantee that each element knows all the network nodes. Therefore, a new node arriving at the network may not receive enough information from the nodes it connects to, if these nodes have not enough capacity to store the addresses of all nodes.

In order to assure that the flooded information is reliable, super-probes are required to update their CKNs, testing some or all addresses, before sending the Pong flooding message. This is called the testing phase (section 4.3.4). Recall that super-probes may receive new addresses from network elements that try to establish connection with it or from the Pong flooding messages it receive. Instead of testing these addresses as soon as they are received, super-probes schedule all tests for immediately before broadcasting the Pong flooding messages. This method is computationally lighter and avoids repeating some tests. Note that in case (i) the testing phase was performed during the connection set-up process. Thus, the Pong flooding process can start immediately after the ending of the network connection phase.

The extent of the CKN update differs in cases (ii) and (iii). In case (ii), which is performed more frequently, the update comprises testing all "non-tested" addresses in the CKN. In case (iii), the CKN is first updated based on the FKN and then, all CKN addresses to which the super-probe is not connected to, whether "tested" or "non-tested", are tested. The update of the CKN based on the FKN follows the procedure described in section 4.3.4 and 4.3.7.6. The procedure of case (iii) allows nodes that have become inactive to be removed from the CKN and nodes of the FKN that have become active to be inserted in the CKN.

After accepting a connection request from a remote node a super-probe may not update its CKN with the address of the remote node in case the CKN is full (section 4.3.7.2.2). As a result, super-probes may not have in their CKN all the nodes to which they are connected to. Due to this, in cases (ii) and (iii) after the testing phase a super-probe should always verify if its LAC contains connections to nodes that are not stored in its CKN. If this is verified and there is now free space available in the super-probe's CKN, it must be filled with the addresses of the identified nodes before the super-probe floods the Pong message.

The rules for flooding Pong messages in the overlay network are described in section 4.2.3. Each network element should update its CKN after receiving a valid Pong message, using the procedure described in section 4.3.7.3.

4.3.6.2 Super-probe demotion negotiation process

The demotion negotiation process is used to keep the number of super-probes in each measurement group at a minimum. In this process two super-probes, of the same measurement group and connected to each other, must determine if both should be maintained in super-probe mode or if one of them can be demoted to the probe mode, based on their available resources (memory occupancy and number of connections that can be accepted) and on the supported security modes. It may be possible that, at a given time, one super-probe (or both) can support all connections maintained by the remote super-probe. In this case, if the super-probe supports all security modes of the remote one, then the remote one can demote itself to the probe mode, after requesting all its elements to reconnect to the surviving super-probe. Conditioning the demotion negotiation process on the set of security modes supported by the demoting super-probe is required because a surviving super-probe can only accept the connections of all elements currently connected to the demoting super-probe if it supports all its security modes.

In the DTMS-P2P system the demotion negotiation process can be performed in two different situations: (i) after the connection set-up process when a super-probe successfully connects to another super-probe of its measurement group or (ii) periodically between a super-probe and each super-probe of its measurement group it is connected to. Case (i) is required because the two super-probes become known to each other for the first time and, therefore, need to verify if one of them has enough resources to support the connections of the other. In this case, the demotion negotiation process is started by the requesting element. Case (ii) is required because, since the network topology can vary significantly, with connections being added and removed frequently, there may be periods of time when the load of a super-probe decreases, making it possible for one of its peer super-probes to support all its connections. In this case, the period used in the demotion negotiation process is configured by the node administrator (numberOfSuperProbeVerificationInterval – 24 hours by default).

In the periodic demotion negotiation process, a super-probe must test only one connection at a time. After testing a connection to a given super-probe, if the requesting super-probe is not demoted to the probe mode and there are more connections to be tested, the super-probe must test the next super-probe of its measurement group. However, in case the requesting super-probe is demoted to the probe mode it must stop the demotion negotiation process.

The demotion negotiation process starts by an exchange of messages, described in section 4.3.6.2.1, where each peer super-probe declares its available resources and if demotion is allowed. Then the super-probes, determine if one of them can be demoted to the probe mode, using the criteria defined in section 4.3.6.2.2.

In case of demotion, the demoting super-probe must first reassign its connections to the surviving super-probe by sending a Bye message (appendix A.2.37), with the address of the surviving super-probe, to all elements connected to it (except to the surviving super-probe). Upon sending a Bye message to a remote element, the demoting super-probe closes the corresponding connection and updates its LAC. The remote element also updates its LAC and tries to connect to the surviving super-probe. After sending all Bye messages, the super-probe should demote itself to the probe mode. Afterwards, the (new) probe must

perform the following actions: (i) clean its Route Table; (ii) stop the Light Data File generation; (iii) stop a network connection process, if it was engaged in one when demoted to probe; (iv) stop any pending demotion negotiation process; (v) send to its super-probe the list of monitoring modules it supports, using the List of Supported Monitoring Modules message (appendix A.2.7) and the list of available Heavy Data Files, using the List of Shared Files message (appendix A.2.8). The (new) probe should also update its FKN based on the CKN because it may contain information not yet stored in the FKN (remember that a super-probe only updates the FKN before processing Pong flooding).

In case of a demotion, the surviving super-probe should update both in its LAC and in its CKN the mode of the connection with the demoting super-probe (section 4.3.7.4).

The super-probes involved in a demotion negotiation process, must process all pending messages, before effectively changing state (demote to probe, for the demoting super-probe, or update the mode of the remote super-probe, for the surviving super-probe). The messages received from the time instant when there was a decision to change state until the time instant when the state was changed effectively must not be processed.

A super-probe may be involved in several demotion negotiation processes at the same time. Thus, before effectively changing to the probe mode, a super-probe must first verify if it is still connected to the remote node, if it is still a super-probe and if the remote one is still a super-probe too. If one of these conditions is not verified, the demotion negotiation process should be stopped. Otherwise, the super-probe initiates a change of state. Moreover, the super-probe must be blocked to perform any other actions while performing the various actions associated with a change of state.

4.3.6.2.1 Messages exchanged during a demotion negotiation process

To start a demotion negotiation process a super-probe must send a Demotion Negotiation Request message (section A.2.9) to a remote super-probe. This message is used to inform the remote super-probe about the number of connections the sending super-probe currently has active, the number of new connections it can still accept, the amount of available memory space the super-probe can still use, the memory space occupied by the sending super-probe's process and if the sending super-probe is configured (or not) to be maintained in super-probe mode. After receiving this message, the remote super-probe should reply with a Demotion Negotiation Response message (section A.2.9). This message has the same fields of the Demotion Negotiation Request message. The Demotion Negotiation Response message must have the same Message ID of the corresponding Demotion Negotiation Request message. The messages exchanged during this process are illustrated in Figure 14.

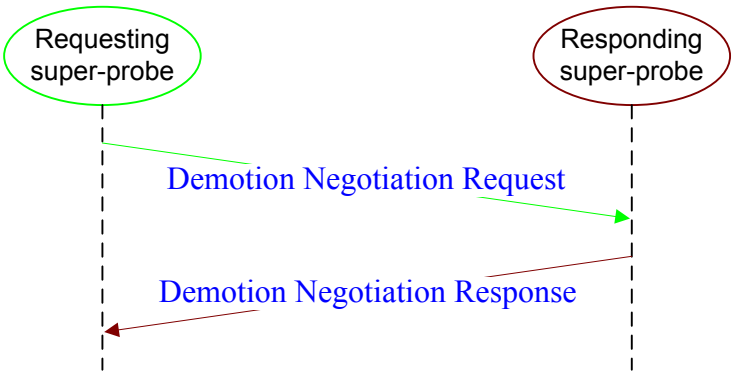

Figure 14. Messages exchanged during the demotion negotiation process.

The responding super-probe must abort the demotion negotiation process and close the connection to the requesting super-probe in the following cases: (i) protocol failure; (ii) the requesting element is not a super-probe of its measurement group; (iii) the responding element verifies that the set of security modes supported by one peer (itself or the requesting element) is not contained in the set of security modes supported by the other. A protocol failure will occur if one or more fields of the received message have invalid values.

The requesting super-probe will abort the demotion negotiation process and close the connection to the remote super-probe in the following cases: (i) protocol failure; (ii) the remote super-probe does not send a Demotion Negotiation Response message during the configured timeout. A protocol failure will occur in the same cases of the responding element; in addition it will occur if the Message ID of the received Demotion Negotiation Response message is different from the one transmitted in the Demotion Negotiation Request message.

If the responding node is no longer a super-probe, it must send the Demotion Negotiation Response message with the mode field of the message header set to “probe”. In this case the content of the message body is irrelevant. In this case the demotion negotiation process is aborted.

After exchanging Demotion Negotiation messages, both super-probes should compare its available resources and decide which super-probe will demote itself to probe mode or if they will both maintain in super-probe mode. This decision is made based on the criteria described in section 4.3.6.2.2.

4.3.6.2.2 Criteria for demoting a super-probe to probe mode

In the DTMS-P2P system, during a demotion negotiation process between two super-probes, the nodes use a unique criterion, known to all nodes, to determine if one can be demoted to the probe mode. As mentioned before, this decision is based on the super-probe’s available resources (free memory and number of connections that can be accepted) and supported security modes.

The demotion of a super-probe will only be considered if one super-probe (or both) supports all the security modes of the other. Otherwise the demotion process will not be started. If the super-probes do not support the same security modes, but one supports all

the security modes of the other, only the later can be considered for demotion. If the super-probes support the same security modes, both of them can be considered for demotion.

A super-probe will not declare to a remote node its total amount of available resources. Instead it will keep a percentage of the total available resources free. This percentage is designated by resourcesFreePercentage (20% default). This margin prevents the super-probe from becoming overloaded, in case it has to hold all the connections of its peer super-probe. Thus, the resources declared are the following:

i. number of available connections (numOfAvailableConnections) = maxNumberOfConnections × (1 - resourcesFreePercentage/100) - number of active connections;

ii. number of available connections to probes (numOfAvailableProbeConnections) = maxNumberOfProbeConnections × (1 - resourcesFreePercentage/100) - number of active probe connections;

iii. amount of available free memory (availableFreeMemory) = maxMemory × (1 - resourcesFreePercentage/100) - amount of free memory, where maxMemory represents the maximum amount of memory the super-probe process can use.

In all three cases, if the obtained value is less than zero it must be set to zero.

In addition to the three parameters described above, a super-probe also declares to its remote node the following parameters:

i. current number of connections to super-probes of the same measurement group (numOfSuperProbeConnections);

ii. current number of connections to super-probes of other measurement groups (numOfSuperProbeConnectionsOtherGroup);

iii. current number of connections to clients (numOfClientConnections);

iv. current number of connections to probes (numOfProbeConnections);

v. amount of memory currently occupied by the super-probe process (occupiedMemory);

vi. if the super-probe is or not configured to maintain the super-probe mode (cannot be demoted to the probe mode).

A super-probe is considered to have enough resources to support the load of a remote super-probe (allowing the remote super-probe to be demoted to the probe mode), only if the three following conditions are verified:

i. the super-probe's availableFreeMemory is greater than or equal to the occupiedMemory of the remote super-probe;

ii. the super-probe's numOfAvailableProbeConnections is greater than or equal to the numOfProbeConnections of the remote super-probe plus one (connection to the remote super-probe if demoted to a probe);

iii. the super-probe's numOfAvailableConnections is greater than or equal to the number of new connections to super-probes (of the same and other measurement groups) it will have to establish plus the number of probe and client connections of the remote super-probe.

In case (i) we consider that the additional memory required by the super-probe in order to support the load of the remote one is equal to the amount of memory occupied by the remote super-probe process.

In case (iii), an approximation is considered for calculating the number of new connections to super-probes, since a super-probe has no means to know what are the super-probes connected to the peer super-probe. We consider that the set of super-probes connected to one super-probe (the one with the lowest number of connections) is contained in the set of super-probes connected to the other. In fact this may not be true, and the super-probe may need to establish additional connections, eventually reaching the maxNumberOfConnections. However, the margin considered in numOfAvailableConnections helps to minimize this problem.

Several situations may occur:

i. The super-probes do not support the same security modes but one of them supports all the security modes of the other and the later is configured to be kept in the super-probe mode. In this case, both super-probes will be maintained in super-probe mode.

ii. The super-probes do not support the same security modes but one of them supports all the security modes of the other and the later is not configured to be kept in the super-probe mode. In this case, only the super-probe that supports the lowest number of security modes is candidate for demotion. It will be demoted only if the peer super-probe can support its entire load, based on the criteria defined above. Note that this verification must be performed by both super-probes, since they both need to determine what the outcome of the verification is.

iii. The super-probes support the same security modes but one is configured to be kept in super-probe mode. In this case, only the super-probe not configured to be kept in the super-probe mode is candidate for demotion. It will be demoted only if the peer super-probe can support its entire load, based on the criteria defined above. Again, this verification must be performed by both super-probes, since they both need to determine what the outcome of the verification is.

iv. The super-probes support the same security modes and both are allowed to be demoted to the probe mode. In this case, each super-probe must verify (i) if it can support the entire load of the remote super-probe and (ii) if the remote super-probe can support its entire load. Here, there are three possible cases:

i. Both super-probes can support the load of the other. In this case, the super-probe with the lowest number of active connections will be demoted to the probe mode. In this way, the number of new connections that the (surviving) super-probe will have to establish is minimized. In case of a tie, the responding super-probe will be demoted to the probe mode.

ii. Only one super-probe can support the load of the other. In this case, the remote super-probe will be demoted to the probe mode.

iii. None of the super-probes can support the load of the other. In this case, both super-probes are maintained in super-probe mode.

4.3.6.3 Load distribution to prevent super-probe overloading

A super-probe accepts only a number of connections from probes (maxNumberOfProbeConnections), to prevent it from becoming overloaded. However, the load that each probe imposes on the super-probe, it is connected to, is not fixed. For example, the super-probe needs to save information about all the files the probe is sharing (section 4.6.1) and this varies a lot from probe to probe. An additional mechanism was introduced in order to cope with this variability.

A super-probe should, after processing each new received message, verify if its free memory space is not below a threshold. This threshold is a percentage (freeMemThresholdPercentage, 5% default) of the maximum amount of memory the super-probe process can use (maxMemory). In case this limit is reached, the super-probe should close a number of connections to probes, until the occupied memory settles below the threshold, again a percentage (resourcesFreePercentage, 20% default) of maxMemory.

Before closing the connection to a probe, the super-probe sends it a Bye message (appendix A.2.37), indicating the cause for closing but without suggesting a new super-probe. A probe receiving this Bye message, will then try to connect to another super-probe using the addresses stored in its CKN. If this is not possible, the probe should try connecting again to the initial super-probe. If this also fails, the probe should promote itself to super-probe mode and reconnect to the network.

Thus, this mechanism redistributes the load of overloaded super-probes to other super-probes and, when this is not possible, creates new super-probes capable of accommodating that load.

4.3.7 Updating the CKN

The updating of the CKN of a network element will be performed in four different situations: (i) during the connection set-up process between any two elements (section 4.3.5), (ii) after receiving a Pong message in the process of Pong flooding (section 4.3.6.1.1), (iii) after a demotion negotiation process (section 4.3.6.2), (iv) after a node disconnect and (v) before the periodic CKN testing phase (section 4.3.6.1.1). In the cases (i) and (ii), the update can be based on the CKN received from the remote element (rCKN) and on the address of the remote element (rAdr). In the cases (iii) and (iv), the update is

only based on the address of the remote node (rAdr). In the case (v), the update is based on the information stored in the FKN.

In cases (i) and (ii), before updating the local CKN, the rCKN is filtered and sorted. The following nodes will be removed from the rCKN: (i) nodes that are already in the local CKN with the same mode and the same measurement group, (ii) nodes that are already in the local CKN with a different mode or measurement group but with a status of "tested", except in the case of a client or probe receiving a Pong flooding message and (iii) nodes from other measurement groups in the case of a client receiving a Pong or a Pong flooding message. Procedure (i) avoids that the RTT measurement is repeated for these nodes. Procedure (ii) was adopted because, since these nodes were already tested, this information is considered to be more reliable than the one of the rCKN. The exception for the case of a client or probe receiving a Pong flooding message is because these elements will no longer test the nodes of their CKN after becoming connected to the network and the information flooded by a super-probe can be trusted since the super-probe always tests all the nodes in its CKN before sending the Pong flooding message. Procedure (iii) was adopted because clients do not attempt connection to nodes of other measurement groups. After these filtering, the RTT is measured for each of the surviving nodes and the rCKN is sorted to adhere to the structure defined in section 4.3.2.

To describe the update of the CKN we will consider separately (i) if and how new nodes can be inserted and (ii) if the attributes (mode and measurement group) of old nodes (nodes already stored in the CKN) can be updated. Regarding the insertion of new nodes there are three update levels:

- Level *n0* – new nodes are not inserted in the CKN
- Level *n1* – new nodes are inserted in the CKN only if there is space available
- Level *n2* – new nodes are inserted in the CKN possibly replacing old nodes

Regarding the attributes of old nodes there are two update levels:

- Level *o0* – the attributes of old nodes are not updated
- Level *o1* – the attributes of old nodes are updated

The update of the CKN may last for a long period of time, in case there are a lot of new nodes to be added to the list. Remember that an element must compute the RTT to all received node addresses to be able to sort them in order of the measured RTTs. Therefore, elements cannot be blocked while updating its CKN. They must keep processing the exchange of messages with remote nodes. Thus, it is possible that, at a given time, an element has more than one update of its CKN scheduled. This may happen, for example, if a node receives connection requests from different nodes at the same time. In this case, the element must process one update at a time, with a FIFO service discipline.

The update of the FKN based on the CKN will be performed in the following situations: (i) after an element completes the network connection process; (ii) by a super-probe, after the periodic testing phase (section 4.3.6.1.1), if there were one or more CKN updates since the

previous period, (iii) by a demoting super-probe, after the demotion negotiation process (section 4.3.6.2) and (iv) by a probe or client already connected to the network, after a CKN update. The update of the FKN is performed as follows: (i) first write all entries of the CKN; (ii) second write the entries of nodes that were in the previous FKN but are not in the CKN. Note that, in this way, the information relative to nodes that were both in the CKN and in the previous FKN is updated, namely, the mode and measurement group. The nodes that were in previous FKN but are not in the CKN must be included again in the FKN because they may become active in the future. Having the FKN updated frequently will help the network reconnection process of an element after a system failure.

4.3.7.1 Overwriting rules

In the case of levels *n2* and *o1* updates, the information regarding nodes already in the CKN may be updated and new nodes are inserted in the CKN possibly replacing other nodes.

The attributes of old nodes will only be updated when the node has a status of "non-tested" in the local CKN or a status of "tested" in the rCKN. This is because the information of a node should only be replaced by more reliable information.

If there is space available in the CKN, a new node will be inserted without removing an old node. Otherwise, if the CKN is full, a new node will be inserted replacing the last old node of a MG selected according to the following procedure:

1. If the MG of the new node is OwnerMG and
    a. its current number of nodes is bellow minOwnerMG (section 4.3.2), select one MG from OtherMGs (see criteria bellow).
    b. its current number of nodes is above minOwnerMG, select OwnerMG.
2. If the MG of the new node belongs to OtherMGs and
    a. its current number of nodes is bellow minOtherMGs, select OwnerMG.
    b. its current number of nodes is above minOtherMGs and
        i. the current number of nodes in the MG of the new node is above maxOtherMGs, select the MG of the new node.
        ii. the current number of nodes in the MG of the new node is bellow maxOtherMGs, select one MG from OtherMGs (see criteria bellow).

To select one MG from OtherMGs, the following criteria is used: first select the largest MG with at least one non-tested probe; if none, select the largest MG with at least one non-tested super-probe; if none, select the largest MG with at least one tested probe; if none, select the largest MG with at least one tested super-probe.

Note that in this case an old node may be removed even if has a higher precedence than the new node. This is because a level *n2* update is only performed when the information received from the remote element (rCKN or rAdr) is considered more reliable.

4.3.7.2 Updating during the connection set-up process

During the connection set-up process, in the handshaking phase (section 4.3.5.2), both the requesting and the responding elements will attempt to update its CKN based on information sent by the remote node, namely (i) the remote CKN (rCKN) and (ii) the address of the remote node (rAdr). The requesting element attempts to update its CKN after receiving a Pong message and the responding element after receiving a Ping message. The way of updating the CKN depends on the type of element and whether or not the responding element is already connected to the network. The latter is indicated in the "Connected" field of the Pong message. The update rules are summarized in Table 1 and in Table 2.

4.3.7.2.1 Updating based on the rCKN

There are five distinct cases regarding the level of update:

- Cases ***a*** and ***d*** – Requesting or responding super-probes perform updates of levels *n1* and *o0* based on the rCKN. This assures that super-probes will test all addresses that are in its initial CKN, when the network connection or reconnection processes start, which improves the network connectivity. One could argue that updates of levels *n2* and *o1* would be more appropriate in the case of super-probes requesting connection to already connected probes or super-probes, because they have more reliable information. However, the adopted solution minimizes the probability of missing information regarding nodes connected to the network: on one side, the super-probe will not lose any of the nodes in its initial CKN, that may have been inserted by the network administrator, since it is performing updates of levels *n1* and *o0* and, on the other side, the super-probe will have access to the nodes listed in the CKNs of other super-probes, when receiving later their broadcasted Pong flooding messages.

- Case ***b*** – A probe or client requesting connection to a node already connected to the network perform updates of levels *n2* and *o1* based on the rCKN. This is because these elements want to connect to the network as soon as possible; they will do so when finding a super-probe of its measurement group. Since the responding element is already connected the network, it may have more recent information that allows to speed-up the network connection process.

- Cases ***c***, ***f*** and ***g*** – Probes and clients requesting connection to non-connected nodes and non-connected probes responding to any element perform updates of levels *n1* and *o0* based on the rCKN. This is because the remote element is not connected to the network and therefore cannot be considered to have more reliable information.

- Case ***e*** – When the responding element is a connected probe its CKN is not updated based on the rCKN (updates of levels *n0* and *o0*). Recall that a probe will no longer test the nodes of its CKN after completing the network connection process. Since the requesting element is in the process of testing the nodes of its CKN, and some of these nodes may not have been tested yet, and if injected in the connected probe could lead to propagation of unreliable information to other nodes. Note that probes will have access to the nodes listed in the CKNs of other super-probes, when receiving later their broadcasted Pong flooding messages.

Note that updates of levels *n2* and *o1* based on the rCKN will never be performed at responding elements since the remote elements are in the process of connecting to the network and, therefore, do not have more reliable information.

When new nodes are inserted in the CKN they are inserted with a status of "non-tested". This is because the received information will be used by the receiving element to connect to those nodes, whenever necessary, and this information may not be completely reliable since the remote element may not have tested all nodes in its CKN.

Table 1 – Rules for updating the CKN of the requesting element based on Pong.

Requesting	Responding	Update Levels / Status		
		rCKN	rAdr	
SP	any	n1 / o0 / nt	o1 / t	***a***
P, C	c SP, c P	n2 / o1 / nt	o1 / t	***b***
	nc SP, nc P	n1 / o0 / nt	o1 / t	***c***

SP – super-probe
P – probe
C – client

c – connected
nc – not connected
t – "tested"
nt – "non-tested"

Table 2 – Rules for updating the CKN of the responding element based on Ping.

Responding	Requesting	Update Levels / Status		
		rCKN	rAdr	
SP	any	n1 / o0 / nt	n1 / o1 / t	***d***
c P	any	n0 / o0	n1 / o1 / nt	***e***
nc P	SP same MG	n1 / o0 / nt	n2 / o1 / nt	***f***
	other	n1 / o0 / nt	n1 / o1 / nt	***g***

4.3.7.2.2 Updating based on the rAdr

If the rAdr is already present in the CKN, the information regarding mode and measurement group is updated when incorrectly set (level *o1* update). Otherwise, a level *n1* or level *n2* update will be performed. These two types of update will only be performed at responding nodes, since requesting elements already have the rAdr in their CKN. There are three distinct cases regarding the update level of nodes not yet present in the CKN:

- Case ***d*** – Responding super-probes perform a level *n1* update based on the rAdr, by the same reasons of the update based on the rCKN.

- Cases ***e*** and ***g*** – Connected probes receiving a connection request from a probe or super-probe and non-connected probes receiving a connection request from a node other than a super-probe of their measurement group perform a level *n1* update based on the rAdr. In both cases, the remote node is in the connection set-up process and, therefore, its mode (probe or super-probe) may still change: a probe can become a super-probe if it cannot connect to a super-probe of its measurement group; a super-probe can become a probe if demoted in the end of the connection set-up process. Note that probes already connected to the network are only allowed to replace entries of their CKN based on Pong flooding messages.

- Case ***f*** – Non-connected probes receiving a connection request from a super-probe of their measurement group perform a level *n2* update based on the rAdr. This is because a probe always rejects connection attempts but, when not connected to the network, will try to do so as soon as possible. Since the remote node is a super-probe of its measurement group, this means the probe has found a potential node to connect to. Therefore, the super-probe should be inserted in its CKN. Note that the super-probe will be inserted in the correct order in the CKN, which depends on the measured RTT; if the measured RTT is smaller than the RTTs of other super-probes of the same measurement group already in the CKN, the new super-probe will be tested immediately; otherwise, it will be tested later (if necessary). The overwriting rules for the level *n2* update are the same as described in section 4.3.7.1.

Note that, in cases ***a***, ***b*** and ***c***, the rAdr must be removed from the CKN of the requesting element if there is a protocol failure or the remote node does not exist.

There are four distinct cases regarding the status assigned to nodes:

- Case ***a***, ***b*** and ***c*** – Requesting elements change the status of the remote node to "tested" after completing the connection set-up phase.

- Case ***d*** – A responding super-probe will set the status of the updated or inserted remote node to "tested" to avoid that the super-probe will test the remote node later which is not necessary.

- Cases ***e*** and ***g*** – Connected probes receiving a connection request from a probe or super-probe and non-connected probes receiving a connection request from a node other than a super-probe of their measurement group set the status of the updated or inserted remote node to "non-tested". This is because probes reject connection attempts from remote nodes and they must test connections to remote nodes by their own when trying to connect to the network. Only after this test a remote address can be considered a "tested" one.

- Case ***f*** – Non-connected probes receiving a connection request from a super-probe of their measurement group set the status of the updated or inserted remote node to "non-tested", by the same reason of the rAdr update level.

If the remote node is a client, its address will not be inserted in the CKN since this list only includes nodes (probes or super-probes).

4.3.7.3 Updating after receiving a Pong flooding

When a node receives a Pong flooding message it will update its CKN based on the received rCKN and on the address of the super-probe that originated the message. The procedure is the same for the two cases.

Given that the Pong flooding message is flooded to all network elements, it may contain the address of the receiving node. In this case, the node must be removed from the rCKN prior to updating the CKN.

When the receiving node is a client or a probe updates of levels *n2* and *o1* are performed. This is because the information flooded by a super-probe can be trusted since super-probes always test all nodes in their CKNs before sending the Pong flooding message. Clients and probes must have this behavior because they can no longer test the nodes of their CKN after becoming connected to the network. Therefore, they must rely on the information received in the Pong flooding messages in order to keep their CKN updated.

When the receiving node is a super-probe again a level *n2* update is performed but with overwriting rules that have some exceptions in relation to the ones described in section 4.3.7.1. In particular, only super-probes can be inserted in a full CKN and only "tested" probes can be removed. Moreover, when the MG to be selected for removal is according to cases 1.a and 2.b.ii of section 4.3.7.1, then the selected MG must be the one with the greatest number of "tested" probes. This procedure guarantees that "non-tested" nodes are never removed, which assures that super-probes will test all addresses that are in its initial CKN. This should be assured for the same reasons of cases ***a*** and ***d*** of the update based on the rCKN (section 4.3.7.2.1). Also, only super-probes can be inserted in the CKN because in order to maximize the connectivity of the overlay network super-probes must keep in their CKNs the highest possible number of (other) super-probe addresses. Finally, "tested" super-probes are never replaced by new super-probes because a super-probe should not risk losing information about a "tested" super-probe stored in its CKN to add another one for which it may not be possible to establish a connection.

In this case, when the receiving node is a super-probe, the attributes of old nodes will only be updated (level *o1* update) when the (local) status is "non-tested". When the (local) status is "tested" we trust the local information; when it is "non-tested" we do not, but will force a test soon.

In the case of clients and probes, the status of inserted or updated nodes must be set to "tested", because they were indeed tested by the super-probe that originated the message immediately before sending it. This will assure that the new nodes received in a Pong flooding message may overwrite the nodes that were tested during the connection set-up process, preventing these (possibly outdated) nodes to be kept indefinitely in the CKN. If these nodes were considered as "not-tested" then the nodes tested during the connection set-up process would never be updated and may never be removed.

In the case of super-probes, the status of inserted or updated nodes must be set to "non-tested", because super-probes must test all received nodes. However, in the case of a super-probe directly connected to the super-probe that originated the Pong flooding message, the address of the later should be inserted with a status of "tested" in the CKN of the former, if that address was not yet present in the CKN.

4.3.7.4 Updating after the demotion negotiation process

In the demotion negotiation process a surviving super-probe must set the mode of the demoted super-probe to "probe" in its CKN (level *o1* update).

4.3.7.5 Updating after node disconnect

When a super-probe detects a loss of connection with a remote node it must change the status of the remote node to "non-tested" in its CKN (level *o1* update). This is because the connection with that node must be tested in the next CKN testing phase to verify if it is still active.

When a probe or client detects a loss of connection with its super-probe, the status of all nodes in their CKNs is changed to "non-tested" and the CKN is updated based on the FKN. These elements will try to reconnect immediately to the network. The update of the CKN based on the FKN is of levels *n1* and *o0*, so new nodes can be inserted only if there is space available and attributes of old nodes are not updated. Note that the FKN is not allowed to overwrite the CKN because the latter has more recent information.

4.3.7.6 Updating before the periodic CKN testing phase

Before the CKN testing phase that occurs at large intervals (30 minutes default), super-probes update the CKN based on the FKN, as probes and clients in previous section.

4.4 Traffic measurements

This section presents the monitoring modules and the mechanisms used to configure traffic measurements and store the measurement results.

4.4.1 Monitoring modules

The monitoring modules are the modules responsible for the actual traffic measurements. Examples are *ping*, *tcpdump* [TCPdump], *tracert* and OWAMP [JOWAMP]. A node of the DTMS-P2P system may support a variety of monitoring modules. When starting a node, the node's administrator must provide it the information about the monitoring modules the node will support and which restrictions must be respected when configuring test sessions using those monitoring modules. A DTMS-P2P node must only allow users to configure test measurements in the monitoring modules it supports.

4.4.1.1 List of supported monitoring modules

In the DTMS-P2P system, a node administrator must configure the monitoring modules the node should support using a XML file. This file will be designated by File of Supported Monitoring Modules (FSMM). We have adopted the XML format due to its readability

which facilitates the exchange of files between different implementations of the DTMS-P2P protocol. The Document Type Definition (DTD) of the FSMM is represented in Figure 15.

```
<!DOCTYPE SupportedMonitoringModules [
  <!ELEMENT SupportedMonitoringModules (monitoringModule*)>
  <!ELEMENT monitoringModule(name, commandToGetHelpDescription, listOfOptionsToSaveToFile, restrictions)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT commandToGetHelpDescription (#PCDATA)>
  <!ELEMENT listOfOptionsToSaveToFile (#PCDATA)>
  <!ELEMENT restrictions (mustUse,doNotUse)>
  <!ELEMENT mustUse (#PCDATA)>
  <!ELEMENT doNotUse (#PCDATA)>
  <!ATTLIST monitoringModule id ID #REQUIRED>
]>
```

Figure 15. File of Supported Monitoring Modules XML DTD.

In this XML DTD the "monitoringModule" element stores the information about a given monitoring module. It comprises the "id" attribute and some child elements. The "id" attribute and the "name" element store the name of the supported monitoring module. The "commandToGetHelpDescription" element describes the command that should be used to get the help description (or usage) of the given monitoring module. For example, for the *tcpdump* monitoring module the command *tcpdump -h* should be the command to be used to get the help description of the *tcpdump* monitoring module. This information can be used by the node to inform a requesting user how a supported monitoring module can be configured (section 4.4.2.4). The "listOfOptionsToSaveToFile" element represents the list of options, separated by ";", that can be used to configure the monitoring module to save the results of a test measurement to a file (for example, the *-w* option of the *tcpdump* monitoring module). This information will be used by the node to identify when a user configured it to store the results of a test session to a file (section 4.4.3). The "restrictions" element represents the restrictions that should be respected by a client, when configuring test sessions to be performed at the respective node. If these restrictions are not followed, the request should be rejected (section 4.4.2.6). The "restrictions" element is composed by two sub elements: the "mustUse" element and the "doNotUse" element. The "mustUse" element is used to define the options, separated by ";", that must be used by a client, when configuring a test session using the respective monitoring module. For example, it can be the option *-n* (Number of echo requests to send) when executing the *ping* command in windows machines. The "doNotUse" element is used to define the options, separated by ";", that must not be used by a client, when configuring a test session using the respective monitoring module. For example, it can be the option *-t* (Ping the specified host until stopped) when executing the *ping* command in Windows machines. The "restrictions" field must be used to prevent the configuration of measurements that may not preserve the privacy of ISPs and users or that might cause harm to the network or to users. These security issues are discussed in [Claffy2006]. Thus, the monitoring modules to be used in the DTMS-P2P system should guarantee the possibility of configuration of security parameters that assure the preservation of user privacy and safe measurements when required. For example, one should be able to define which information can be captured and reported in passive measurements or limit the number of packets to be sent to the network in active measurements.

Figure 16 shows an example of a FSMM. In this example, the node is configured to support four monitoring modules: *ping*, *tracert*, *owping* and *tcpdump*.

```
<?xml version="1.0"?>
<SupportedMonitoringModules>

  <monitoringModule id="ping">
    <name>ping</name>
    <commandToGetHelpDescription>ping</commandToGetHelpDescription>
    <listOfOptionsToSaveToFile></listOfOptionsToSaveToFile>
    <restrictions>
      <mustUse>-n 10</mustUse>
      <doNotUse>-t</doNotUse>
    </restrictions>
  </monitoringModule>

  <monitoringModule id="tracert">
    <name>tracert</name>
    <commandToGetHelpDescription>tracert</commandToGetHelpDescription>
    <listOfOptionsToSaveToFile></listOfOptionsToSaveToFile>
    <restrictions>
      <mustUse></mustUse>
      <doNotUse></doNotUse>
    </restrictions>
  </monitoringModule>

  <monitoringModule id="owping">
    <name>owping</name>
    <commandToGetHelpDescription>owping -h</commandToGetHelpDescription>
    <listOfOptionsToSaveToFile>-F; -T</listOfOptionsToSaveToFile>
    <restrictions>
      <mustUse>senderPort 4181; receiverPort 22368; -P 21164-21174</mustUse>
      <doNotUse></doNotUse>
    </restrictions>
  </monitoringModule>

  <monitoringModule id="TCPDUMP">
    <name>TCPDUMP</name>
    <commandToGetHelpDescription>tcpdump -h</commandToGetHelpDescription>
    <listOfOptionsToSaveToFile>-w</listOfOptionsToSaveToFile>
    <restrictions>
      <mustUse>-c; -i 2</mustUse>
      <doNotUse>-F; -l; -m; -r</doNotUse>
    </restrictions>
  </monitoringModule>

</SupportedMonitoringModules>
```

Figure 16. Example of a File of Supported Monitoring Modules.

All the nodes of the DTMS-P2P network should be configured to support, at least, the *ping* and Trace Route monitoring modules. The Trace Route monitoring module has different names in Windows (*tracert*) and Linux (*traceroute*) Operating Systems.

4.4.1.2 Sending of information regarding supported monitoring modules

In the DTMS-P2P system, a super-probe must keep the list of monitoring modules that each probe connected to it supports. This information is centralized in super-probes to save in signaling messages when clients query the DTMS-P2P system about the nodes that support a given monitoring module. Thus, a probe must send the information about which monitoring modules it supports to the super-probe it connects to, right after completing its network connection process (section 4.3.4). This information is sent in a List of Supported

Monitoring Modules message (appendix A.2.7). In this message the probe sends a list of the hash codes of the monitoring modules' names it supports, converted to lower case (appendix A.3). Using the hash codes of the monitoring module's name, smaller messages are generated. Hash codes are also used because the super-probe's memory will be used more efficiently since each word hash code only occupies 4 bytes (the name of the monitoring modules may occupy more bytes).

The List of Supported Monitoring Modules message must only be sent by a probe if it supports one or more monitoring modules (Figure 17).

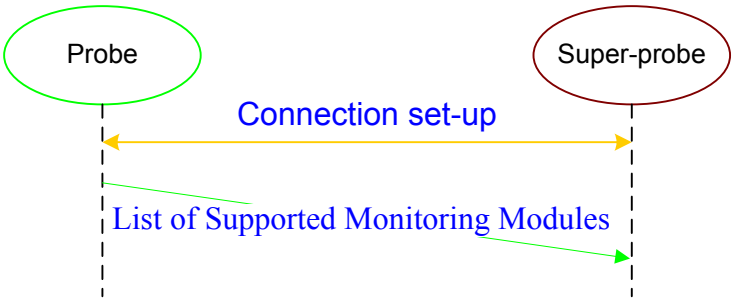

Figure 17. Sending of information regarding supported monitoring modules.

After receiving this message a node must verify if it is a super-probe. If not, the message must be discarded and the connection where the message was received must be closed. Only super-probes can receive this type of message.

The List of Supported Monitoring Modules message may be received more than once. If the super-probe verifies that the message was already received before, from the sending probe, it should replace the list of supported monitoring modules related to the sending probe with the information received in the new message. This situation may happen because a running probe may have its configuration changed, to support a new monitoring module or to not support a given monitoring module anymore. In these cases, the probe should inform the super-probe to which it is connected of this change, by sending a new List of Supported Monitoring Modules message to it.

4.4.2 Measurement tests configuration

Performing a measurement in the DTMS-P2P system requires indicating the address and measurement group of the implicated nodes and the name of the monitoring module. If a user knows this information in advance, it can enter it manually in the DTMS-P2P client. For other cases, the client includes several features to help users selecting the nodes and modules where the measurements will be performed:

1. A user may know the monitoring module but not the measurement nodes. Instead of maintaining a list of all active network nodes, which could be expensive in terms of memory occupancy, the client maintains a list of all active measurement groups. After selecting the measurement group, the user may configure the client to ask for a list of all nodes from a specific measurement group that support a specific measurement module. The user is also given the possibility to configure the client to ask for a list of all network nodes. However, as referred above, it is not advisable to use this option since it could be very expensive.

2. A user may know the measurement group but neither the address of the measurement nodes nor the monitoring module. In this case, the user may configure the client to ask for the list of all nodes from a specific measurement group and, after receiving this list, to ask for a list of monitoring modules supported by a specific node.

3. A user may know the measurement nodes (address and measurement group) but not the monitoring module. In this case, the user may configure the client to ask just for a list of monitoring modules supported by a specific node.

After the selection of the measurement nodes and of the monitoring module, the client shows the user the list of restrictions associated with the selected monitoring module. Since a user may not know exactly how to configure the module, the user may configure the client to ask for the usage description of that module.

The protocols used by the client to prepare and execute a test session are described in the following sections.

4.4.2.1 Measurement group discovery request

This section explains how a client obtains a list of all measurement groups in a DTMS-P2P system. The client must send a Measurement Group Discovery Request message (appendix A.2.10) to its super-probe (Figure 18). This message should then be flooded among all super-probes, as described in section 4.2.3. Each super-probe receiving this message, must reply with a Measurement Group Discovery Response message (appendix A.2.11), where it indicates the Group ID of its measurement group. This message is forwarded back to the client using the path taken by the request message, as described in section 4.2.3.

The super-probe to which the client is connected doesn't need to send a Measurement Group Discovery Response message, because the client already knows the super-probe and its Group ID.

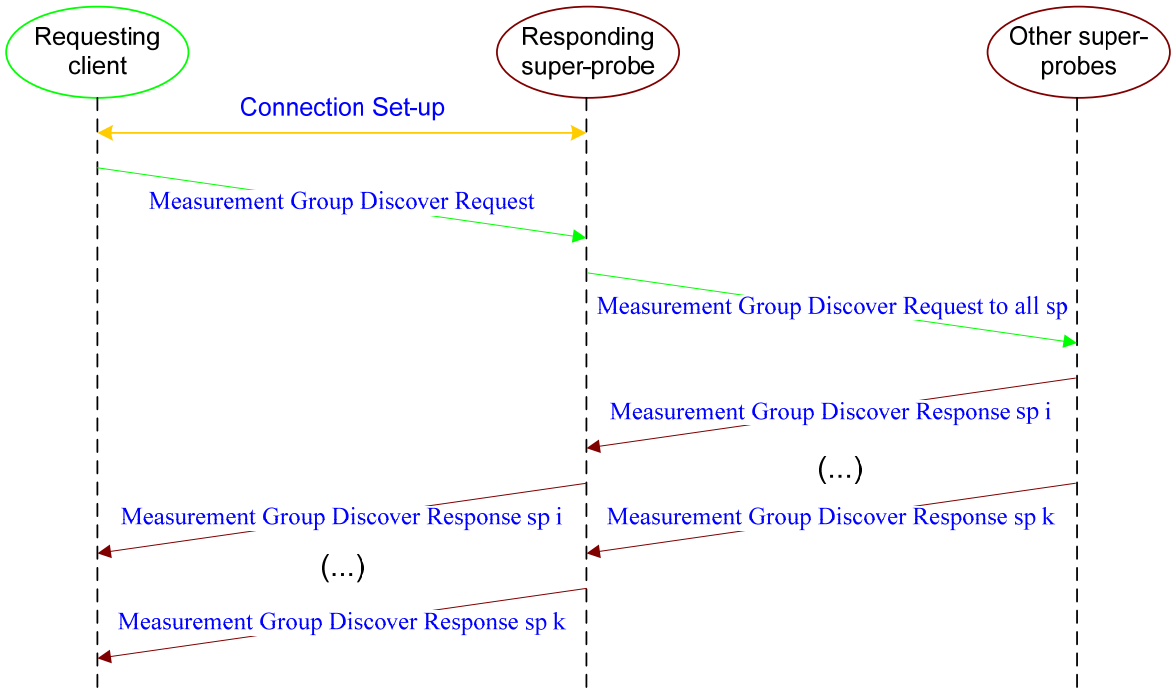

Figure 18. Messages exchanged during the measurement group discover process.

With all the Measurement Group Discovery Response messages received in response to the Measurement Group Discovery Request message, the client can build its list of known measurement groups of the network. The maximum length of this list should be configured by the client's administrator. This list may have two fields: the Group ID of the measurement group and the number of super-probes of that measurement group.

The request is sent by the client upon completion of the connection set-up process. In addition, the request is sent whenever the user prompts the client to do so. Recall that the characteristics of the DPMS-P2P network can vary frequently, with new measurements groups being added and old measurement groups being removed from the network. Clients may also be configured to periodically send this request.

4.4.2.2 List of nodes retrieval

A client may request information on a list of nodes. There are three cases: request (i) the list of all nodes from a specific measurement group that support a specific monitoring module, (ii) the list of all nodes from a specific measurement group and (iii) the list of all network nodes. In all cases the information can be retrieved from super-probes.

4.4.2.2.1 List of all nodes from a specific measurement group that support a specific monitoring module

To obtain the information about all nodes of a specific measurement group that support a specific monitoring module, the client must send a List of Nodes Discovery Request message (appendix A.2.12) to its super-probe, indicating the Group ID of the measurement group and the name of the monitoring module. This message is then flooded among all super-probes, as described in section 4.2.2. Each super-probe of the measurement group indicated in the request message (and only those), must reply with a List of Nodes

Discovery Response message (appendix A.2.13) (Figure 19). In this message the responding super-probe identifies all its probes that support the requested monitoring module. It can also include itself if it also supports the requested monitoring module. This message should only include nodes that support a common security mode with the client. This is because only in this case will the client be able to connect to the node and retrieve the results of a configured test session (section 4.2.4). The security modes supported by the client are indicated in the request message (as in any message). A super-probe receiving a request message does not need to forward it to its probes, because super-probes maintain information on the monitoring modules and security modes supported by all its probes. The List of Nodes Discovery Response message should be forward back to the client using the path followed by the request message, as described in section 4.2.2. A response message should not be sent when the super-probe has no positive answer to the request.

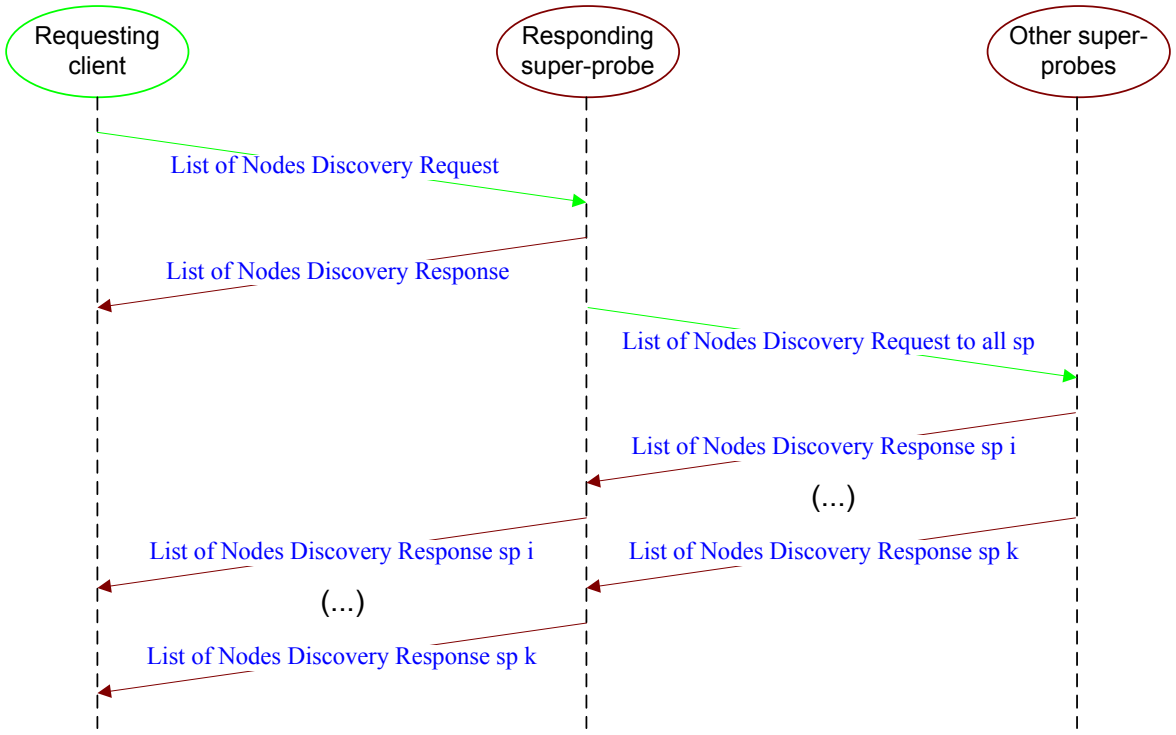

Figure 19. Messages exchanged during the list of nodes retrieval process.

After receiving a List of Nodes Discovery Response message, the client should present to the user the received information and this information should be updated each time a new List of Nodes Discovery Response message is received. The user can then choose the nodes to be used in the test session. If the client does not receive the first response message within a configured timeout (section 4.2), then it should inform the user about this event. All List of Nodes Discovery Response messages, received after the user has chosen the nodes to be involved in the test session should be discarded. To do so the client must remove the record corresponding to the request from its list of sent messages (CLM – section 4.2).

The information received in the List of Nodes Discovery Response messages should be stored in a temporary file and not in the client's device memory because a lot of nodes may support the requested monitoring module. The content of the file should be the information presented to the user.

4.4.2.2.2 List of all nodes from a specific measurement group

To obtain the information about all nodes of a specific measurement group, the client must send a List of Nodes Discovery Request message (appendix A.2.12) to its super-probe, indicating the Group ID of the required measurement group. This message must not contain the name of a monitoring module as in the case described in section 4.4.2.2.1. This message is then flooded among all super-probes, as described in section 4.2.2. Each super-probe of the measurement group indicated in the request message (and only those), must reply with a List of Nodes Discovery Response message (appendix A.2.13) (Figure 19). In this message the responding super-probe identifies itself and all its probes. The List of Nodes Discovery Response message should be forward back to the client using the path followed by the request message, as described in section 4.2.2.

After sending the List of Nodes Discovery Request message, the client must wait for responses and process them as described in section 4.4.2.2.1.

4.4.2.2.3 List of all network nodes

To obtain the information about all network nodes, the client must send a List of Nodes Discovery Request message (appendix A.2.12) to its super-probe. This message must not comprise a measurement group ID neither the name of a monitoring module as in the case described in section 4.4.2.2.1. This message is then flooded among all super-probes, as described in section 4.2.3. Each receiving super-probe must reply with a List of Nodes Discovery Response message (appendix A.2.13) (Figure 19). In this message the responding super-probe identifies itself and all its probes. The List of Nodes Discovery Response message should be forward back to the client using the path followed by the request message, as described in section 4.2.3.

After sending the List of Nodes Discovery Request message, the client must wait for responses and process them as described in section 4.4.2.2.1.

4.4.2.3 List of supported monitoring modules retrieval

To obtain the list of monitoring modules a given node supports, the client must send a List of Supported Monitoring Modules Request message (appendix A.2.14) to its super-probe, indicating the Group ID of the node's measurement group and the node's address (Figure 20). Then, the message should be flooded to the destination node, as described in section 4.2.1. After receiving this message, the destination node must reply with a List of Supported Monitoring Modules Response message (appendix A.2.15). In this message the responding node must include the information about all the monitoring modules it is configured to support and the restrictions that must be respected when configuring test sessions with them, as was configured by its administrator (section 4.4.1.1). This message is forwarded back to the client using the path taken by the request message, as described in section 4.2.1.

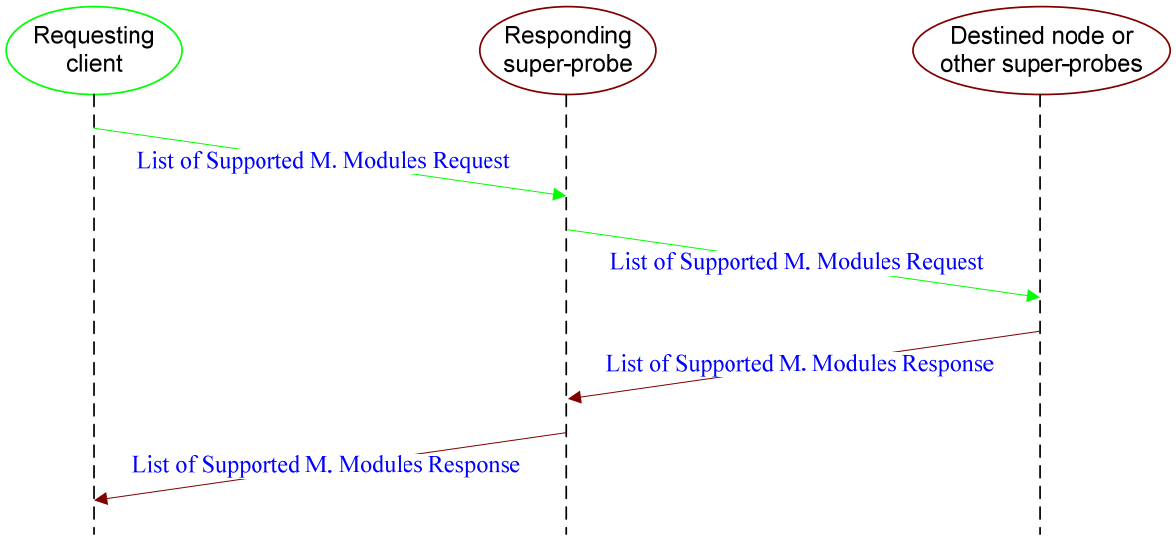

Figure 20. Messages exchanged during the list of supported monitoring modules request process.

After receiving the List of Supported Monitoring Modules Response message, the client should provide the user with the list of monitoring modules the remote node supports. The user can then choose the module he wants to use to configure the test session among the monitoring modules the remote node is configured to support. If the client does not receive a response message within a configured timeout (section 4.2), it should stop waiting for the response and should inform the user about this event.

4.4.2.4 Monitoring module's help retrieval

To retrieve the help or usage description related to a given monitoring module a remote node supports, the client must send a Monitoring Module Help Request message (appendix A.2.16) to its super-probe, indicating the node's measurement group and address and the name of the required monitoring module (Figure 21). This message should be flooded to the destination node, as described in section 4.2.1.

If the node receiving the Monitoring Module Help Request message is a super-probe and it verifies that the message is destined to a node (probe or super-probe) connected to it, it should forward the message to the destination node even if it verifies that the remote node does not support a common security mode with the requesting client (section 4.2.4). In this case, the receiving super-probe could be configured to not forward the request to the destination node, since the requesting client does not support a common security mode with the destination node (section 4.2.4). However, the super-probe should forward the message anyway because the user may only want to get the help or usage description of the required monitoring module and is not willing to configure test measurements at the remote node. But, if the destination node is a probe connected to it, before forwarding the Monitoring Module Help Request message, the receiving super-probe should first verify if the probe supports the required monitoring module. In case this condition is not verified the super-probe should reject the request (Monitoring Module Help Response message – appendix A.2.17). In this case, the receiving super-probe does not need to forward the Monitoring Module Help Request message to the probe because it will reject the request too. Note that the security modes supported by the client are indicated in the request message (as in any message) and super-probes maintain information on security modes

supported by all elements connected to them and information on the monitoring modules supported by all its probes.

After receiving the Monitoring Module Help Request message, the destination node must respond with a Monitoring Module Help Response message (appendix A.2.17). In this message, and if it supports the required monitoring module, the responding node must include the help or usage description related to the requested monitoring module. If the node does not support the required monitoring module, it should reject the request. The Monitoring Module Help Response message must be forward back to the client using the path followed by the request message, as described in section 4.2.1.

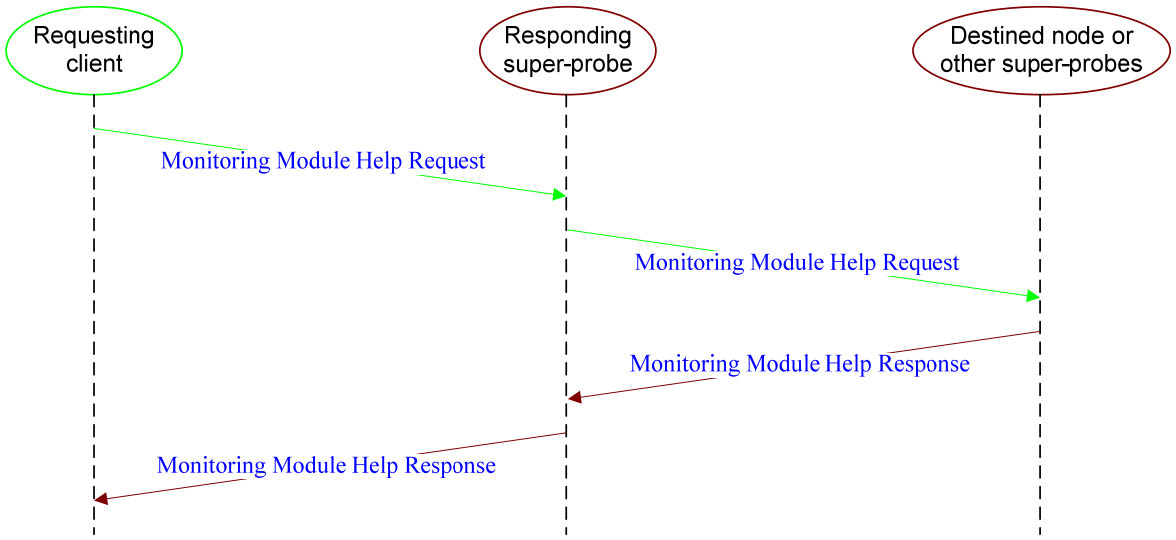

Figure 21. Messages exchanged during the monitoring module help request process.

After receiving the Monitoring Module Help Response message, the client should present to the user the received help or usage description related to the required monitoring module. Using this information, the user is able to configure the test measurements he is willing to execute. If the client does not receive a response message within a configured timeout (section 4.2), it should stop waiting for the response and it should inform the user about this event.

4.4.2.5 Monitoring module's list of restrictions retrieval

To get the list of restrictions of a monitoring module from a specific node the client must send a Monitoring Module List of Restrictions Request message (appendix A.2.18) to its super-probe. This message should then be flooded to the destination node, as described in section 4.2.1. After receiving this message a super-probe must verify if the message is destined to a node (probe or super-probe) connected to it. If this is the case, the super-probe should verify if the remote node supports a common security mode with the requesting client (section 4.2.4), before forwarding the received message to it. In case this condition is not verified the super-probe should reject the client's request. In addition, if the destination node is a probe under the control of the receiving super-probe, the super-probe should also verify if the remote probe supports the required monitoring module. In case this condition is not verified the request must be rejected too. In case of rejection, the receiving super-probe should send a Monitoring Module List of Restrictions Response

message (appendix A.2.19) to the requesting client, rejecting the request. In this case, the receiving super-probe does not need to forward the Monitoring Module List of Restrictions Request message to the destination node connected to it because it will reject the request too.

After receiving the Monitoring Module List of Restrictions Request message, the node to which the message is destined must verify if it supports the monitoring module in the message and if it has a common security mode with the requesting client (section 4.2.4). Then, it should send a Monitoring Module List of Restrictions Response message informing the requesting client if it accepts or not the request (Figure 22). A node only accepts the request if it supports the monitoring module in the message and if it has a common security mode with the requesting client (section 4.2.4). If the node accepts the request, the Monitoring Module List of Restrictions Response message must comprise the list of restriction the node is configured for the given monitoring module. This message should be forward back to the client using the path followed by the request message, as described in section 4.2.1.

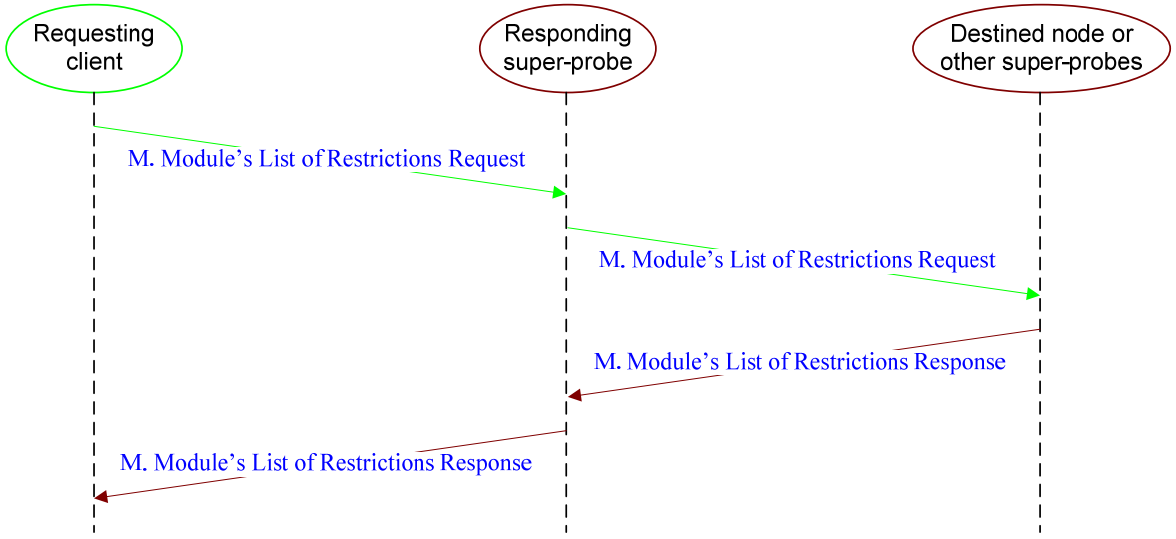

Figure 22. Messages exchanged during the monitoring module's list of restrictions request process.

After receiving the Monitoring Module List of Restrictions Response message and if the request was accepted, the client must present to the user the information about the restrictions that must be respected, when configuring test sessions at the node he chose. If the client does not receive a response message within a configured timeout (section 4.2), it should stop waiting for the response message and it should inform the user about this event. The test session configuration must also be aborted, if the receiving node rejects the request.

4.4.2.6 Test session configuration and execution

After choosing a node where a test session should be processed and with the information about the restrictions that apply to the selected monitoring module, the user is able to configure a test session using the DTMS-P2P client.

4.4.2.6.1 Messages exchanged during a test session configuration

The configuration of measurement tests is accomplished by sending a Command message (appendix A.2.20) destined to the node where the measurement test should be performed. In this message the client indicates the node's measurement group and address plus the monitoring module and all configurations parameters that should be used to configure the test session (command line with the required options, e.g. *ping –n 10 www.ua.pt* for a node running in a Windows machine). The client sends the Command message to its super-probe. This message is then flooded among all super-probes, as described in section 4.2.1.

After processing a received Command message, the destination node may send a response to the requesting client (Figure 23). The response to the Command message is the Command Response message (appendix A.2.21). With the Command Response message the node can inform the client, if it accepted and successfully processed or not the command received in the Command message. If the command was successfully processed, the Command Response message should comprise the name of the file where the results of the test session were stored. Otherwise, the Command Response message should comprise the description of the error (a descriptive text) occurred while processing the command. It is up to the user to determine if the remote node should send the Command Response message or not. Thus, after the user introduces the command line to be executed at the remote node, the client must ask him if he wants to wait or not for the response to the requested command. The Command message to be sent to the remote node must be configured according to the user willingness. The destination node will only send the Command Response message, if the user chose to wait for this message.

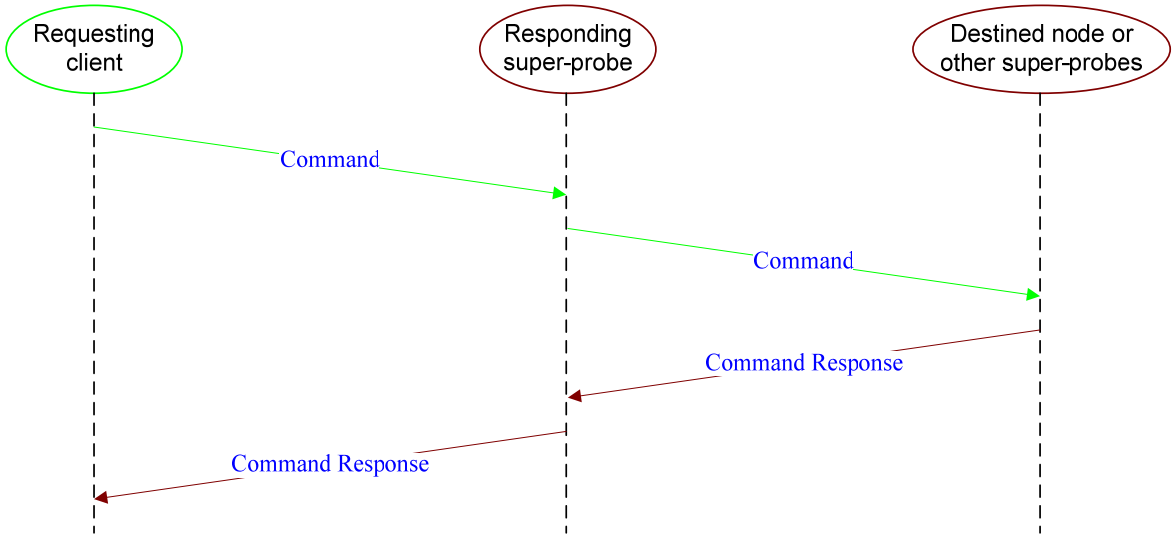

Figure 23. Messages exchanged during the command request process.

After sending the Command message to its super-probe and if the user chose to wait for the response to this request, the client must wait for the response until the response is received or the user chooses to stop waiting for it. Thus, in this case, the waiting time is not determined by the timeout associated with request-response interactions because, if the remote node accepts the request, it will only send the response after the end of the configured test session and some test sessions may last longer than this timeout.

If the user didn't choose to wait for the response to its request, he should be able to configure new test sessions, while the previous one is being processed.

After receiving a Command message destined to a node connected to it, a super-probe must process the same verifications relative to the supported security modes and required monitoring module as described for the case of a Monitoring Module List of Restrictions Request message (section 4.4.2.5). In case of rejection, the super-probe should send a Command Response message to the requesting client, if the user configured the client to wait for a response.

The test session configuration must be aborted, if the receiving node rejects the request. This may happen, (i) if the destination node does not support the requested monitoring module, or (ii) does not support a common security mode with the requesting client (section 4.2.4), or (iii) the user did not respect all the restrictions configured for the required monitoring module at that node, or (iv) the destination node has no more space to store the results of new test sessions and or (v) an error occurred while executing the requested command.

The case (iii) occurs if the user did not use a required option or if the user used an option that shouldn't be used.

The case (iv) may happen if the node's results directory has reached its maximum size or if there is no more free space in the node's local machine storage unit. The node's results directory is the directory where all test results of test sessions processed by the node must be saved. The location of this directory and the maximum size that this directory can reach should be defined by the node's administrator, when starting the node. The node's administrator may not define a size limit for this directory, in which case the node can store information in it as along there is free space in the node's storage unit.

4.4.2.6.2 Test session execution

After receiving a Command message and if the node is able to process the requested command, it must execute it in a separate process. The node must not be blocked while processing the command because it should always be listening for incoming messages and for incoming connections.

The selected node must only start processing the requested command at the startime configured by the requesting user. This information is received in the Command message. A startime in the past means that the node must immediately start processing the requested command. This is another advantage of the DPMS-P2P system because it allows a network administrator to define which monitoring module to be used to configure a test measurement, where the measurement should be performed and additionally when the measurement should be carry out.

4.4.3 Storing the measurement results

The results of all test sessions processed by a node must be saved to the results directory defined by the node's administrator. In the DTMS-P2P system, the file with the results of a test session (Heavy Data File – HDF) must have a name composed by the Group ID of the

measurement group of the client from where the request was received, the client's IP address and port number, the Group ID of the measurement group of the node where the test measurement was performed, the IP address and port number of the respective node, the timestamp corresponding to the time when the test was started (measured in milliseconds since 0h on 1 January 1970) and the command line used to configure the test session. The extension of the file should be ".res". Thus, the following format must be used:

Client Group ID_Client IP Address.port_Node Group ID_Node IP Address.port_timestamp_Command Line.res

In case the numeric representation of the IP address version 6 is used, all ":"characters should be replaced with the "." character. File names should not include the ":" character. Thus, an example of the results file name may be 00000000000000000000000000000000_192.168.0.4.21164_00000000000000000000000000000000_192.168.0.1.22368_1175895448445_ping 192.168.0.2 -n 200.res.

Some monitoring modules may have options to save the results directly to a file. For example, *tcpdump* may be configured to save the results to a file, using the option *-w*. Thus, before processing the command, the node must verify if an option to save the results to a file is in the requested command. Therefore, for each supported monitoring module, the system local administrator must provide the node the information of all options to save results to a file the monitoring module provides (section 4.4.1.1). If an option to save the results to a file is found in the requested command, the node must redirect it to its DTMS-P2P results directory and use the file name format described above. Thus, an example of the results file name of a test session configured using the *tcpdump* monitoring module may be 00000000000000000000000000000000_192.168.0.4.21164_00000000000000000000000000000000_192.168.0.2.22368_1175895498746_tcpdump -c 10 -i 2 -w tcpDump.cap.res, if the user chose the string "tcpDump.cap" as the name of the file were the results should be stored. To redirect the results of the configured test measurement to the DTMS-P2P results file presented above the node must execute the following command: *tcpdump -c 10 -i 2 -w <path to DTMS-P2P results file>*, where *<path to DTMS-P2P results file>* is the complete path to the DTMS-P2P results file.

If no option to save the results to a file is used in the received command, it means that the output of the command execution must be saved to the DTMSP2P result's file.

While executing the requested command, the node must always verify if the maximum size configured for the result's directory has not been reached. If yes, the node must stop the test session execution. In this case, the current results in the result's file must be kept.

A user may configure a node to start executing more than one test at the same monitoring module and at the same time in the future. For example, a user may configure twice the same node to process a ping to a given IP address starting at the same instant in the future. Due to this fact, the name of the files, where the results of the configured commands will be stored, will be exactly the same. To prevent this situation, a node must never start executing the required commands exactly at the same time. The node should wait for a few milliseconds between start executing the respective command lines. Having this behavior

the node will guarantee that the files with the results of the requested commands will have different names due to the time instant it starts executing them.

If an error occurs while executing a requested command, the node must store the output of the command execution, which should describe the occurred error. Thus, it is supposed that all monitoring modules a node may support, should output the description of errors that may occur during the execution of the monitoring module. This description should be stored into a file in the error directory. This directory should be, by default, inside the results directory defined by the node's administrator. The file where the description of the error is saved, should have the same name the result file would have, but with the ".err" extension. This file should be created because the node should send it to a requesting client, whenever the remote client requests for the results of the test session that originated it. In this case too, if the node is supposed to send a Command Response message to the requesting client, the node must add to the message the error description of the occurred error. In this way, the user will be able to get the information about the error that occurred during the execution of the requested command. An error may occur in the following situations:

i. The monitoring module is not installed or not accessible. This may happen when the node's administrator configured a node to support a given monitoring module, but it is not installed or not accessible;

ii. There is an error in the requested command. For example a bad option is in the list of arguments of the requested command. For example, if a user requests the following command to a node: *ping -d www.ua.pt*. The *ping* command in Windows does not support the option *-d*. For instance, the following error description should be send in the Command Response message and stored in the error file:

Bad option -d.
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [[-j host-list] | [-k host-list]] [-w timeout] target_name
Options:

-t	*Ping the specified host until stopped. To see statistics and continue - type Control-Break; To stop - type Control-C.*
-a	*Resolve addresses to hostnames.*
-n count	*Number of echo requests to send.*
-l size	*Send buffer size.*
-f	*Set Don't Fragment flag in packet.*
-i TTL	*Time To Live.*
-v TOS	*Type Of Service.*
-r count	*Record route for count hops.*
-s count	*Timestamp for count hops.*
-j host-list	*Loose source route along host-list.*
-k host-list	*Strict source route along host-list.*
-w timeout	*Timeout in milliseconds to wait for each reply.*

At the end of the command execution, if the node is a probe and it successfully generated a Heavy Data File, it must send to its super-probe the information of the new Heavy Data File using the List of Shared Files message (appendix A.2.8). This message should be sent to the probe's super-probe even if the new Heavy Data File contains the description of an error occurred during the execution of a requested command.

If the node successfully performed the requested test session and is configured to replicate the Heavy Data Files it generates, it should start the file replication process to replicate the new HDF (section 4.5).

4.5 File replication

4.5.1 Overview

The DTMS-P2P system uses a distributed data archive system. The nodes store locally the results of their measurements in the Heavy Data Files, but may also replicate these files in other nodes of the same (preferably) or other measurement groups. This approach stands between two major archive philosophies: centralized and completely distributed. Replicating the data in some nodes guarantees that data remains available even if the node that made the measurements becomes inactive. Having multiple data sources also allows a faster and more reliable data retrieval using multi-source download techniques.

Replication will take longer and will occupy more disk space for larger files. By default a Heavy Data File should be replicated at least in one location. The node's administrator may be given the option to decide if the maximum number of replications (maxNumOfReplications) (i) should be or (ii) should not be defined as a function of the file size. Moreover, in case (i), the node's administrator may define the number of replications for each range of file sizes. For example, replicate files greater than 1 GB only once, files between 500 MB and 1 GB twice, files between 100 MB and 500 MB 5 times and files smaller than 100 MB 10 times. In case (ii), the node's administrator only needs to define the maximum number of replications to be performed.

The process of replicating a file from a source node to a receiving node is performed using HTTP. Thus a TCP connection must be established between these two nodes. The TCP connection may fail if the node that waits for the TCP connection request (TCP SYN) is behind a firewall. This node is called a firewalled node. In order to circumvent this problem, the DTMS-P2P includes a feature that allows the TCP connection to be started on either the source node or the receiving node. There are four cases:

i. None of the two nodes is firewalled – in this case the receiving node opens the TCP connection and sends the request for file download;

ii. The source node is firewalled but the receiving node is not – in this case the source node opens the TCP connection and asks the receiving node to use the TCP connection for file download;

iii. The receiving node is firewalled but the source node is not – same as case (i).

iv. Both nodes are firewalled – in this case replication is not possible.

To process the required number of replications of a new Heavy Data File, the (original) source node must first search for possible locations where the file can be replicated. There are two possible scopes for the replication search process. The search can span over all super-probes (global scope) or only over the nodes of the (original) source node measurement group (local scope).

The procedure used in file replication is the following:

1. In order to search for potential storing nodes where a Heavy Data File can be replicated, the (original) source node (probe or super-probe) sends a request message indicating the file name, the file size, the maximum number of potential storing nodes to be returned (maxNumOfStoringNodes), the scope of the replication process, if firewalled nodes should or not be returned and if it is or not firewalled. The maximum number of potential storing nodes to be returned should be equal to the maximum number of replications that still need to be performed. When the replication process is started, this number is equal to maxNumOfReplications. This is because the (original) source node is not able to determine a priori if it will or not receive more than one response. The request message is flooded to all super-probes. If the (original) source node is a probe the request message is sent to its super-probe and, afterwards, is flooded to all super-probes.

2. Upon receiving a request message, each super-probe that match the replication search scope, try to find among its probes a number maxNumOfStoringNodes of probes that match the criteria for file replication, using the procedure of section 4.5.3.1.2. The super-probe will then respond to the request message with the list of probes where the file can be replicated. The super-probe itself can also be included in this list, if the number of identified probes is less than maxNumOfStoringNodes and it matches the criteria for file replication (section 4.5.3.1.4). For each node in the list the following information is supplied: download speed, available disk space, and if the node is firewalled or not. The (original) source node will sort this list according to download speed first, available disk space second and if node is firewalled or not third. If the (original) source node is a super-probe, while sending the request message, it will also try to find among its probes a (small) number (numOfReplications, 2 by default) of probes that match the criteria for file replication, using the procedure of section 4.5.3.1.2. This limit should be defined to guarantee that the super-probe does not replicate the file only at the probes under its control, in case there are other possible locations where the file can also be replicated.

3. When a response from a super-probe is received and if the maximum number of replication requests has not been reached, a (small) number of replications requests (numOfReplications) to nodes listed in this message is started; the nodes selected are the first ones from the sorted list. After this, the content of the response message (with the selected nodes removed from the list of potential storing nodes) is placed in a circular list together with the response messages from other super-probes. The message must be placed in the end of this list. However, if the message's list of potential storing nodes becomes empty, the message must be discarded.

4. When a response from a super-probe is received and if the maximum number of replication requests has been reached, the content of the response message is placed in the beginning of the circular list together with the response messages from other super-probes.

5. After completing a file replication, a receiving node must send a confirmation to the (original) source node. When a confirmation is received (positive or negative), if the maximum number of replications requests has not been reached and there are still potential storing nodes, again a (small) number of replication requests is started. The selected nodes are the first ones from the sorted list of the first response message in the list of response messages. If there are not enough potential storing nodes in the first message, then the subsequent ones will be used. After this, the selected nodes are removed from their lists of potential storing nodes. If a list is not empty, the associated response message is placed in the end of the list of response messages; otherwise it is discarded.

6. When a confirmation regarding the replication of a file at a receiving node (positive or negative) is received, if the maximum number of replications has not been reached but there are not enough potential storing nodes, then a new request for potential storing nodes is flooded to all super-probes.

7. When a positive confirmation is received and if the maximum number of file replications is reached, the file replications process must be stopped.

8. In order to control the replication process we define time periods, which are relatively long (24 hours by default), and will be called replication periods. The (original) source node must try to create all replicas of the file during a replication period. If it is not able to complete the replication of a file during a replication period, then new replication periods should be started. There is a maximum number of replication periods (maxNumOfReplicationPeriods, 4 by default). When a new replication period starts, all the requests made in the previous period and for which a confirmation was not yet received must be discarded. After this, the (original) source node starts a (small) number of replication requests if there are potential storing nodes in the list of response messages or sends a request for potential storing nodes otherwise. Within a sequence of replication periods, the first half will have a local scope while the second half will have a global scope; within a replication period all request messages have the same scope.

In Figure 24 we give an example of a file replication process. We will assume that the original source node was configured to create seven replicas of each file in a maximum of four replication periods. To simplify the example we will consider that each replication request will trigger only one replication (numOfReplications equal to one). At time t_0 the original source node floods a search request to the overlay network with local scope, trying to find potential nodes where the file could be replicated. Latter, at times t_1 and t_2 it receives responses from two super-probes, SP1 and SP2, respectively. We will assume that SP1 and SP2 listed in their search response messages three and two potential storing nodes, respectively. Thus, at times t_1 and t_2 the original source node will send replication requests to the first potential storing nodes indicated by SP1 and SP2. At time t_3, the confirmation of the request issued at time t_1 is received, which drives a new replication request, now sent to the second potential storing node indicated by SP1. The confirmation of this request is received at time t_4, which again drives a new replication request sent to the last potential storing node indicated by SP1. During this replication period no more confirmations are received, in particular those in response to requests issued at times t_2 and t_4. Thus, at the end of the first replication period, at time t_5, only two replicas where successfully

performed (2 replicas in the network). At this time, the original source node still has potential storing nodes for which replication request was not issued, namely the last node indicated by SP2. This request is now sent and its confirmation is received at time t_6. Since all known potential storing nodes have been tried, the original source node floods a new search request, with local scope. A response is received at time t_7, say from super-probe SP1, and a replication request is immediately issued. We will assume that this response only listed one potential storing node. The confirmation to this request is received at instant t_8 and a new potential storing nodes search request, with local scope, is flooded to the overlay network. However, no responses are received until the end of the second replication period, at time t_9. Thus, at the end of the second replication period only two more replicas where successfully performed (4 replicas in the network). At this time, a new search request is flooded to the overlay network, but now with global scope. Latter, at times t_{10}, t_{11} and t_{12} the original source node receives responses from three super-probes, SP4, SP5 and SP6, respectively. We will assume that all super-probes listed in their search response messages only one potential storing node. Thus, at times t_{10}, t_{11} and t_{12} the original source node sends replication requests to the potential storing nodes indicated by the three super-probes, respectively. At times t_{13} and t_{14}, the confirmations of the requests issued at times t_{10} and t_{11} are received. Both drive to two new potential storing nodes search requests. At time t_{15} the original source node receives a response from SP7 to the request sent at time t_{14}. We will also assume that SP7 listed in its search response message only one potential storing node. This response drives a new replication request. No responses were received for the request issued at time t_{13}. During this replication period no more confirmations are received, in particular those in response to requests issued at times t_{12} and t_{15}. Thus, at the end of the third replication period, at time t_{16}, only more two replicas where successfully performed (6 replicas in the network). At this time, a new search request is flooded to the overlay network, again with global scope. Latter, at time t_{17} the original source node receives a response from super-probe SP8. With the information received in this response the original source node sends the last replication request which confirmation is received at time t_{18}. Thus, the replication process only completes at the fourth replication period, at instant t_{18}, when the last confirmation is received.

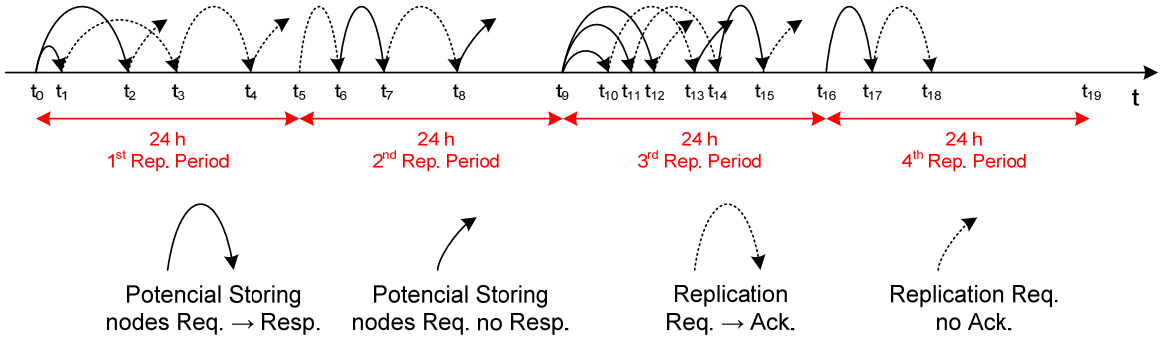

Figure 24. Replication periods.

An exception to procedure 1 occurs in the following case. If the source node is a probe and requests only for not firewalled locations to replicate a file (because it is behind a firewall), its super-probe will not forward this message and will not try to find possible locations where the file can be replicated among the probes under its control, if it is not firewalled and it is a possible location to replicate the file. In this case, the super-probe should return only its own address in the response message. The requesting probe will first replicate the

file in its super-probe and only then will restart the replication search process for new potential storing nodes. In this new process, the probe can now search for firewalled nodes too, since these nodes will be able to download the file from the replica already stored in the super-probe. This must be done to speed-up the file replication process and to guarantee that the file may be replicate at firewalled locations even when the (original) source node is firewalled. This exception is not mandatory since it does not compromise the protocol interoperability.

In the procedure 2, when a super-probe is to be included itself in the list of potential storing nodes, it should do so even if it is busy (all its slots are taken – section 4.7.1). This will allow replicating the file at a later time when the super-probe becomes available. Note that a file will not be replicated at a busy probe; therefore, when all probes under the control a super-probe are not potential storing nodes (section 4.5.3.1.2), this procedure represents a last opportunity to replicate a file within that network zone.

Procedures 1 to 6 aim at spreading the replicas by nodes that are geographically away from each other, trying to avoid the concentration of many replicas in nodes under the control of a single super-probe. If replicas were all concentrated in nodes under the control of a single super-probe, and the network zone where the super-probe is located becomes inaccessible, then the measurement results would become completely unavailable. Recall that probes always try to connect to the closest super-probe; thus it is likely that probes under the control of the same super-probe are geographically near to each other, while super-probes are likely to be geographically away from each other. In order to spread the replicas geographically, the (original) source node should try to distribute them by nodes under the control of as many super-probes as possible. However, since the (original) source node cannot determine in advance how many responses it will receive and when they will be received, (a few) replications must be started as soon as a new response (from a new super-probe) or a new confirmation regarding a replication is received.

According to procedure 8 a number of replication periods is allowed. This has several advantages. First, it allows the scope of the search to be enlarged from local to global if it is not possible to place all the replicas in the measurement group of the (original) source node. Second, it allows recovering from situations where the state of the (original) source node changes during the replication process, e.g., when the (original) source node is a probe and loses connection with its super-probe or when the (original) source node is a super-probe and is demoted to the probe mode. If these cases occur, the responses to requests will not be received by the (original) source node. Giving the opportunity to have new replication periods will allow the (original) source node to perform the remaining replications. In the case of a probe that lost connection with its super-probe, if the probe gets connected to a new super-probe before the end of the replication period is reached then the replication process should proceed as usual; otherwise, the number of replication period will not be incremented until the probe gets connected to a new super-probe.

According to procedure 8 the first replication periods will have a local scope whereas the last ones will have a global scope. This is because it is preferable that files are replicated in the measurement group of the (original) source node, since this speeds-up the file search and download processes by a client connected to the (original) source node's measurement group (local search – section 4.6).

When a file is successfully replicated at a node, this node can be used later as a new source from where the file can be retrieved. The (original) source node maintains a list of nodes where the file was successfully replicated (storingNodesTable) and this list is supplied when a replication request is sent to a replicating node. Based on this list, the replicating node will try to retrieve the file from multiple sources (multi-source download – section 4.7.3). The following restrictions apply: (i) when both the (original) source node and the replicating node are firewalled, only non-firewalled nodes will be selected for multi-source download; (ii) if the file has already been replicated in one or more nodes belonging to the measurement group of the replicating node then, whenever possible, the file should only be retrieved from these nodes; otherwise, the file should be retrieved from all available source nodes. Note that there are other strategies that can be implemented without compromising the interoperability between nodes, e.g., retrieve from all nodes where the file is replicated irrespective of measurement group. Our strategy has the advantage of concentrating the load in one measurement group (alleviating other parts of the network), but may not be optimal in terms of download time (from multiple sources).

4.5.2 Replication Table and Replication Record

A node should be able to replicate more than one file at the same time. To be able to control this process, the (original) source node should maintain a table, called Replication Table, with some information related to each individual file replication process (Replication Record).

Each Replication Record must have a unique ID (Record ID), used to identify the record in the Replication Table. This unique ID should be equal to the hash code (appendix A.3) of the name of the file to be replicated. This unique ID will also be used to identify received messages related to the replication of a given file. The length of the Replication Table should be configurable and the addition of new records to it should be circular – when the maximum number of records that can be saved in the Replication Table is reached, the older record must be replaced with the new one. Before starting the replication process of a new Heavy Data File, the source node must create a Replication Record to be used during the file replication process.

The Replication Record comprises the following fields:

- The name and size in bytes of the file to be replicated;
- The Replication Record's ID (Record ID);
- A flag indicating if firewalled locations can be chosen for file replication. If the (original) source node is not firewalled or the file has been successfully replicated in a not firewalled location, firewalled locations can be chosen as possible locations for file replication. Otherwise, only not firewalled locations should be chosen;
- The maximum number of replications (maxNumOfReplications) to be performed (section 4.5.1);

- The number of replication requests that have been sent and for which a negative confirmation was not received (numOfReplicationRequests). When a negative confirmation is received this number should be decremented. Additional replications can only be requested when this number is lower than maxNumOfReplications.

- The total number of replicas that have been successfully completed (totalNumOfReplications). When this number equals to the configured maxNumOfReplications, the replication process should be stopped;

- A list of the received response messages with information about possible locations where the file can be replicated (listOfPotentialStoringNodesDiscoveryResponses – section 4.5.1). This list should not be greater than maxNumOfReplications, to prevent from overloading the memory in case too many response messages are received.

- The number of replication periods which stores information about in which replication period the replication process is (numOfReplicationPeriods – section 4.5.1);

- A list with the information about other nodes from where the file can be retrieved, besides the (original) source node (storingNodesTable). These nodes are nodes where the file has been successfully replicated before (section 4.5.1). This is a list of lists, indexed first by measurement group and second by node. For each node, the list includes information on its IP address, firewalled state and upload speed.

4.5.3 Messages exchanged in the file replication process

The messages exchanged between a (original) source node and the other nodes of the network, to try to produce the required number of replicas of a given Heavy Data File, are illustrated in Figure 25.

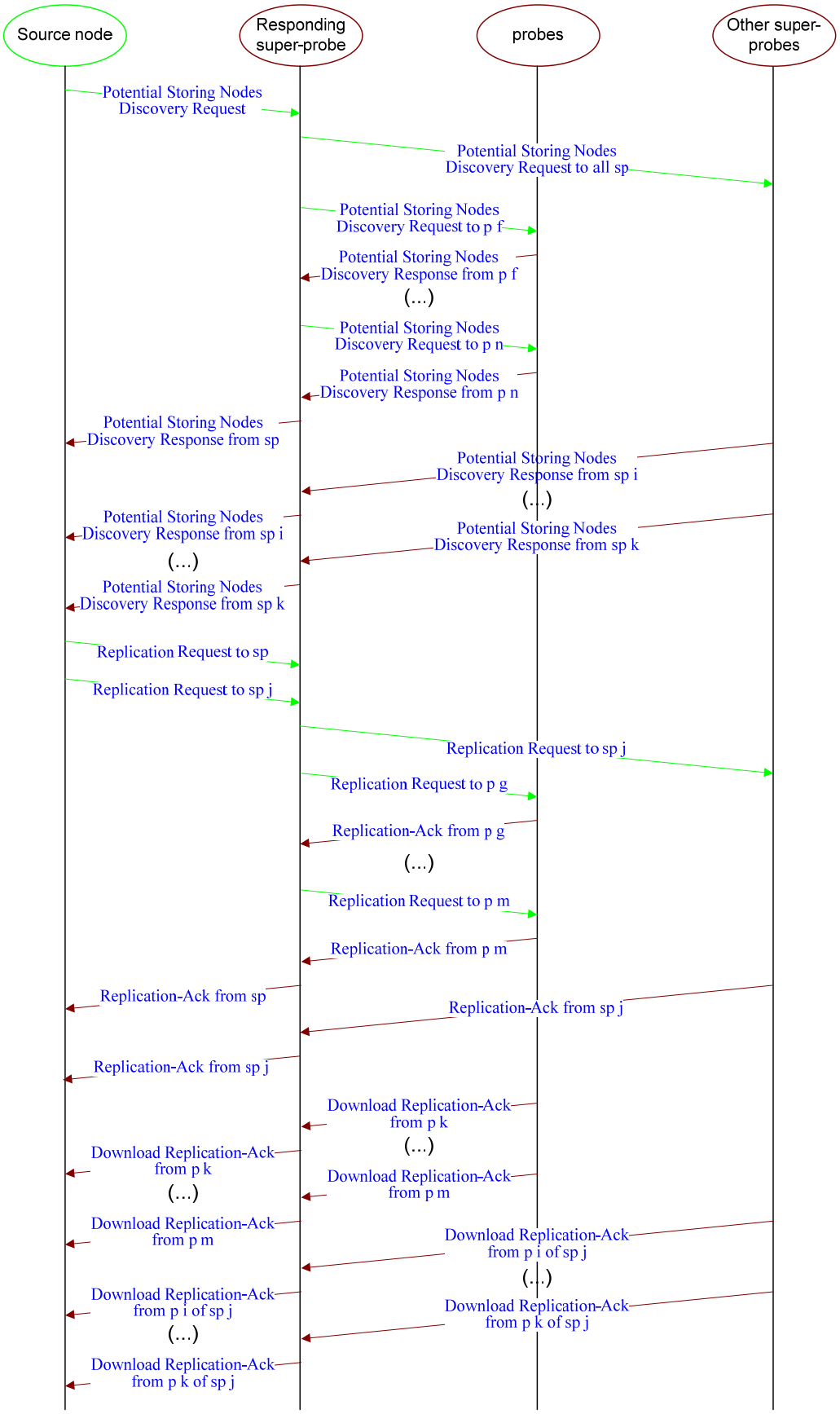

Figure 25. Messages exchanged during the replication process.

As can be verified in Figure 25, there are two types of interactions in the file replication process, one for searching possible locations for file replication and another for replicating a file at a node. Both interactions will be detailed in the next sections.

4.5.3.1 First interaction

4.5.3.1.1 Potential storing nodes discovery

To initiate a search process, the (original) source node (probe or super-probe) sends a Potential Storing Nodes Discovery Request message (appendix A.2.22) that should be flooded to all super-probes, as described in section 4.2.3.

After receiving a Potential Storing Nodes Discovery Request message, a super-probe that matches the search scope of the message should try to find, among the probes under its control, the required number of locations for file replication, as described in section 4.5.3.1.2. If a super-probe finds one or more possible locations, it should answer to the requesting node with a Potential Storing Nodes Discovery Response message (appendix A.2.23) identifying these locations. This message must be forwarded back to the source node using the path taken by the request message, as described in section 4.2.3.

A node receiving a Potential Storing Nodes Discovery Response message must first confirm if it has a record in its Replication Table with an ID equal to the Record ID field of the received message (section 4.5.2). If a match is found, the information received in this message should be used to complete the file replication process. If a match is not found the message must be discarded; it may be a response message related to a replication process already finished. However, in case the node is a super-probe and has been a transit node for the corresponding request message, then the response message must be forwarded back to the source node as described section 4.2.3. In this last case the super-probe must have a record correspondent to the Message ID of the request message in its Route Table (section 4.2).

4.5.3.1.2 How super-probes discover possible locations to replicate a file among the probes under their control

A super-probe needs to discover possible locations for file replication among its probes, when it is the (original) source node or when it receives a Potential Storing Nodes Discovery Request and it matches the message replication search scope.

In order to discover possible locations for file replication among its probes, a super-probe will send them, one by one, Potential Storing Nodes Discovery Request messages. After sending one Potential Storing Nodes Discovery Request message the super-probe will wait during a configured timeout interval for a Potential Storing Nodes Discovery Response (section 4.5.3.1.3). This process is repeated until the super-probe identifies a number of locations where the file can be replicated, equal to two times the number of locations that the super-probe needs to find. In order to assure that the set of tested probes is not always the same, to avoid overloading some probes, the list of probes (taken from the LAC) is scanned circularly and the first probe to be tested is selected randomly. The best among the probes that were identified as possible locations for file replication will be selected; this is accomplished by sorting the list of selected probes according to download speed first,

available disk space second and firewalled state third (not firewalled probes will be selected first).

A super-probe should only test probes that support a common security mode with the (original) source node. This should be done because if both nodes do not support a common security mode, they will not be able to interconnect to process the file replication (section 4.2.4).

If the probe (i) does not share a common security mode with the (original) source node or (ii) does not match the requested replication search scope as indicated in the Potential Storing Nodes Discovery Request message, then there was a protocol failure and the probe should close the connection with the super-probe.

A probe is considered to be a possible location for replicating a file if (i) it is not busy (a probe is busy when all its slots are taken – section 4.7.1) and (ii) it matches the criteria for file replication (section 4.5.3.1.4). In case (i), the probe should not accept the request because it is not possible to determine for how long it will remain busy. Thus, in these cases, the file should be replicated at other nodes not currently busy, which may improve the replication process.

In order to accept a replication request, a probe should add information about its available resources to the Potential Storing Nodes Discovery Response message. An empty Potential Storing Nodes Discovery Response message means that the probe does not accept the replication of the file in its storage space.

If the super-probe receives a response message accepting the replication of the file but indicating the probe is in a firewalled state that does not match the requested one, then there was a protocol failure and the connection with the probe should be closed.

To determine the best locations to replicate a file, a super-probe will not test all probes under its control; it will only test twice the number of locations where the file needs to be replicated. We believe that this is a good trade-off between time to find file locations and optimally in selecting file locations. However, this criterion may vary among different implementations without compromising interoperability.

4.5.3.1.3 Super-probe behavior after sending a request message to a probe under its control

After sending one of the possible request messages used in the replication process to a probe under its control, a super-probe must not send any other messages to the probe until it receives the required response message from it. In case the expected response message is not received after a configured timeout, the super-probe must close the connection to the remote probe. While waiting to receive the response message, the super-probe must process any other messages received from the remote probe or from any other element connected to it. These messages must be processed in parallel. A super-probe must have this behavior because, before the probe sends the response message to it, other messages may be sent by the probe whenever a change occurs (e.g. List of Shared Files, List of Supported Monitoring Modules, etc) and, it is also possible that, during this waiting interval, the super-probe receives messages from other elements connected to it.

4.5.3.1.4 Potential storing node

A node is considered to be a possible location for replicating a file if (i) it matches the requested firewalled state, (ii) it supports a common security mode with the (original) source node, (iii) it has enough free storage space to store the file, (iv) it is not already sharing a file with the same name of the file to be replicated (a previous replication of the file may already have been processed) and (v) it has not the temporary file used to store the file to be replicated (to download a file, a node should first reserve the needed storage space in a temporary file, which will be used to save the file to be downloaded – section 4.7.1).

While calculating its free storage space in (iii), the node, if configured to do so, must account for the percentage of storage space that should be kept free (20%, default). This percentage should be calculated over the (configured) maximum size of the node's results directory; however if no maximum size is configured, then the percentage should not be reserved. Reserving a percentage of storage space to be kept free will prevent the node to enter in a state where all the storage space of its results directory is used by file replications.

4.5.3.2 Second interaction

4.5.3.2.1 Replication request

After selecting one or more nodes for file replication, the source node starts the actual replication process by sending a Replication Request message (section 4.5.3 and appendix A.2.24), indicating the addresses of all nodes where file replication should be performed. If the source node is a super-probe then the destination of a request message may be a probe under its control or another super-probe, from where a Potential Storing Nodes Discovery Response message was received. Otherwise, if the source node is a probe then the destination for the request message is a super-probe, from where a Potential Storing Nodes Discovery Response message was received. The request messages are forwarded in the overlay network as described in section 4.2.1.

After receiving a Replication Request message, a node must first verify again if it is still a possible location for replicating the file (section 4.5.3.1.4). If not, the request must be rejected. If yes, the node should reserve the required space to store the file in a temporary file, until the actual download is performed. The name of this temporary file may be, for example, equal to the name of the file to be replicated, including extension, but with the extension ".tmp". The download process will overwrite the temporary file with the file to be replicated. After completing the download, the file must be renamed to the (original) file name.

Nodes that receive a Replication Request message should reply with a Replication-Ack message. This message indicates how many replication requests were accepted by the replying node. A (original) source node verifies that a Replication-Ack message is in response to a Replication Request message that it has sent based on the Record ID of the Replication-Ack message which should match one of the Replication Record IDs in the Replication Table.

A super-probe which is an (original) source node will send the request message to its selected probes one-by-one: it sends a Replication Request message to one probe, waits for a Replication-Ack or a timeout (section 4.5.3.1.3), and only after that proceeds to the next probe. When the Replication-Ack message is received indicating that the replication request was accepted, the numOfReplicationRequests field of the Replication Record is incremented by one. In case one (or more) probes do not accept to replicate the file, the super-probe must determine how many more replications request are needed to complete the maxNumOfReplications. Then, the super-probe must perform these additional requests (section 4.5.1).

An (original) source node, must update the numOfReplicationRequests field of the Replication Record immediately after sending a Replication Request message to a remote super-probe; it is incremented with the number of nodes for file replication indicated in the request message. If, upon receiving the corresponding Replication-Ack message the source node verifies that not all replication requests were accepted then it decrements the numOfReplicationRequests field accordingly; it must then determine how many more replication requests are needed to complete the total number of replications (maxNumOfReplications) and perform the additional requests, using the procedure described in section 4.5.1.

When a Replication Request message is destined to a super-probe (i) it may have addresses of probes under the control of the destination super-probe and/or (ii) it may have the address of the super-probe itself. In case (i) the receiving super-probe must then send a Replication Request message to the required probes, in the same way described above for the (original) source super-probe case. These messages are essentially a copy of the Replication Request message, except that they are destined to the replicating probe and should not include the list of replicating nodes (which is irrelevant in this case). The messages include the address and Group ID of the (original) source node, which is required for sending the Download Replication-Ack message (section 4.5.3 and appendix A.2.26) to the (original) source node, which notifies if the file replication at the node was successful or not. The super-probe should construct the Replication-Ack message to be sent to the (original) source node based on the information received in the Replication-Ack messages received from the various probes under its control and based on its own potential for becoming a storing node, if prompted to do so. Then, this message should be forwarded back to the (original) source node using the same path of Replication Request message, as described in section 4.2.1.

4.5.3.2.2 File replication and replication confirmation

If a node accepts the replication request received in a Replication Request message, it must download the Heavy Data File from one or more locations where the file is stored. To choose the nodes from where the file should be downloaded, the receiving node must use the information about the nodes storing it, received from the (original) source node in the Replication Request message. Using its measurement group ID and knowing if it is firewalled or not, the node is able to determine where is/are the best location(s) from where the file can be downloaded (section 4.5.1).

Before starting the file download, the node must increment by one the number of download slots in use. If all the node's download slots are in use, it should wait until it has a free slot

to download the file to be replicated. The file download is processed using HTTP, as explained in section 4.7.

After the file download process, the node must send a Download Replication-Ack message (appendix A.2.26) to the (original) source node to inform it if the node successfully replicated or not the file. This message is flooded back to the (original) source node, as described in section 4.2.1.

After sending the Download Replication-Ack message, if the sending node is a probe and it successfully replicated the file, it must also send to its super-probe the information about the new file using the List of Shared Files message (section 4.6.1 and appendix A.2.8).

After receiving a Download Replication-Ack message destined to it, the (original) source node must verify if it has, in its Replication Table, a record corresponding to the file indicated in the received message. If a record is not found, the node should discard the received message: it may be related to an already finished replication process. If a record is found, the node must verify from the received message if the file was successfully replicated. If yes, the node must update the Replication Record's storingNodesTable and the totalNumOfReplications field (section 4.7.1.2). If the total number of replications that should be made (maxNumOfReplications) has been reached, the node must remove the Replication Record from its Replication Table and stop the replication process. However, if the file was not successfully replicated, the number of replication requests made (numOfReplicationRequests – section 4.7.1.2) must be decremented.

As described in section 4.5.1, in case the (original) source node is a probe and it is firewalled but its super-probe is a possible location to replicate the file and it is not firewalled, then the file should be first replicated at the probe's super-probe. In this case, after receiving the Download Replication-Ack message and if the file is successfully replicated at the remote super-probe, the probe must restart the replication search process to find other possible locations (firewalled or not) where the file can be replicated to perform the remaining needed number of replications.

After processing the received Download Replication-Ack message, if the maxNumOfReplications was not reached yet, the node should start another (small) number of replication requests as described in section 4.5.1. If the file was successfully replicated in a not firewalled node, nodes behind a firewall may now be used as possible locations to replicate the file.

4.6 Results search

In this section we will describe how the test measurements results can be searched in the network. But first we will describe how the probes inform their super-probes which files they are sharing to the network since this information is used in the results search mechanism.

4.6.1 List of shared files

In the DTMS-P2P system, a super-probe must keep information about the Heavy Data Files (HDFs) stored in each probe connected to it. This information is centralized in super-

probes to save in signaling messages when clients query the DTMS-P2P network about existing HDFs. It is kept in the list designated by probeListOfSharedFiles. It is used when a client searches the results of configured test measurements or requests the replication and/or deletion of a HDF (sections 4.6.2 and 4.9.1). A probe must send to its super-probe information about its HDFs (i) after completing its network connection process (if one or more files are available at this time) and (ii) after assembling a new HDF.

The information is sent by probes in a List of Shared Files message (appendix A.2.8) and relates to the names of the HDFs (Figure 26). More specifically, each HDF is represented by a number of hash codes from the keywords that make up the file name. Before computing the hash codes the keywords are converted to lower case. The use of hash codes helps saving memory since each word only occupies 4 bytes. The keywords are obtained by breaking up the name of the file on any non-alphanumeric characters (anything but letters and numbers). A space is the standard separator between words. But the separator between words may also be the following characters: "_" and "-". In case there are repeated keywords to be sent in a List of Shared Files message, only one of them is actually sent.

Figure 26. List of Shared Files.

After receiving a List of Shared Files message a node must verify if it is a super-probe. If not, the message must be discarded and the connection where the message was received must be closed. Only super-probes can receive this type of message.

Based on the information received in the List of Shared Files message the super-probe must update its probeListOfSharedFiles. The update is such that repeated keywords are not kept in the list. In this way, the amount of stored information can be much less, when compared with the case of storing the complete name of all files, since many files may have the same keywords in their names. This of course comes at the price of lower efficiency in the query process, since a probe may not have a HDF that matches simultaneously all the keywords indicated by a user.

4.6.2 Results search mechanism

In the DTMS-P2P system, the results of configured test measurements are stored at the node where the test was performed in a so-called Heavy Data File (HDF) which is possibly replicated at other nodes of the network. The HDFs can be searched using a mechanism similar to the one used in Gnutella 0.6 [Gnutella].

Two kind of results search can be performed in the DTMS-P2P overlay network. A client connected to a super-probe of a particular group can perform global and local searches. A

global search is performed in all groups of the monitoring system while a local search is restricted to a given measurement group chosen by the user.

A client initiates a search by sending a Query message (appendix A.2.27) to its super-probe. Each receiving super-probe should process the message and forward it to the other nodes of the network, as described in section 4.2.3. After processing the message, a receiving node may answer with a Query-Hit message (appendix A.2.28) with information about files that satisfy the query. The messages exchanged during the results search process are illustrated in Figure 27.

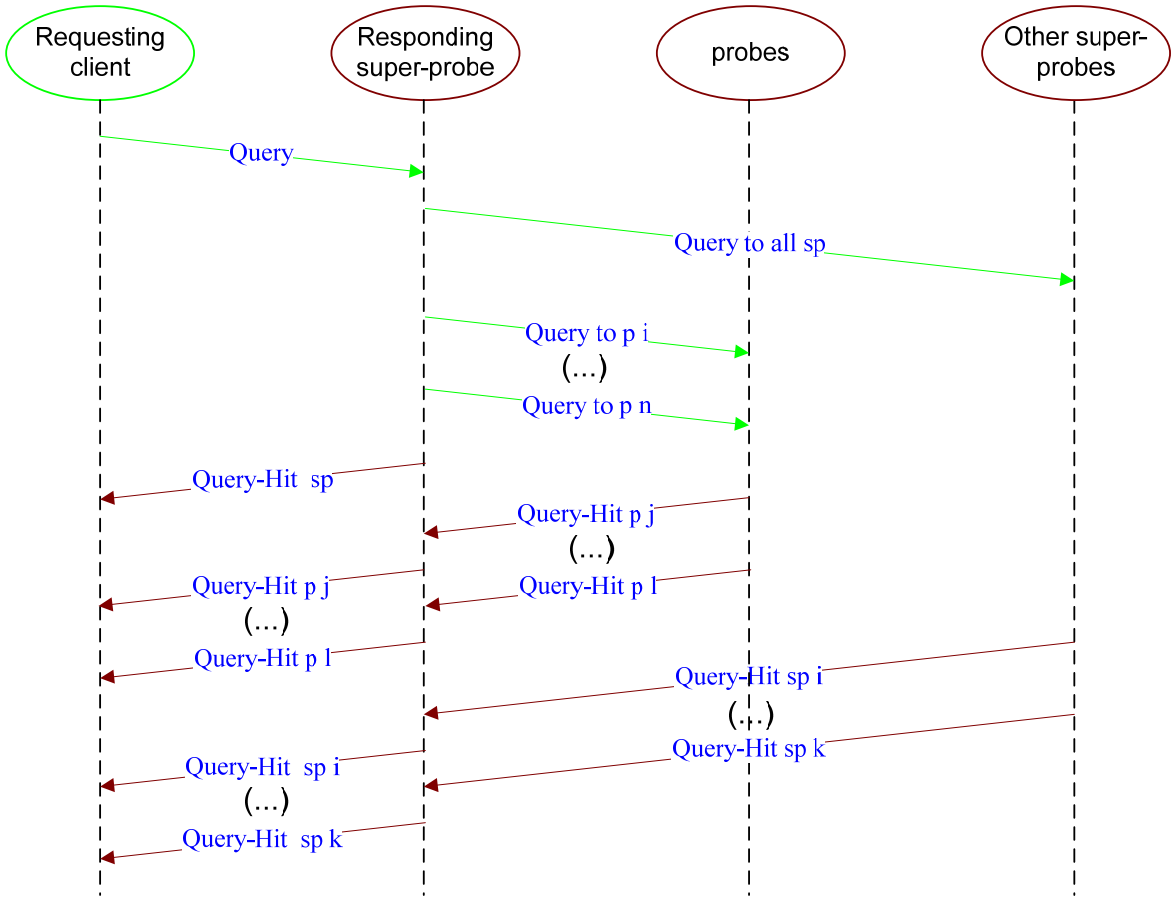

Figure 27. Messages exchanged during the results retrieval process.

The Query message includes the search criterion used to identify files the client is looking for. The search criterion may be the name of the file to be searched or a set of keywords. The Query message has also information about the minimum upload speed the receiving node should support in order to respond to the request. A node receiving a Query message, requesting a minimum speed of *N* Kbits/s, should only respond with a Query-Hit message, if it satisfies the search criterion and it is able to communicate at a speed greater or equal to *N* Kbits/s.

After receiving a Query message a node must verify if the Query message search scope is global or local. If the Query message search scope is global the node should process it. If the Query message search scope is local the node must first verify if the message is

designated to its measurement group, and only in this case should the message be processed.

Before processing a Query message, a super-probe should forward the message to other nodes connected to it. This improves the response process by reducing the time the user has to wait for responses, because all nodes quickly receive the Query message.

When a super-probe matches the search scope of a Query message, it will also forward the message to its probes that may be sharing the files that satisfy the message's search criterion. For each one of its probes, the super-probe has a list of hash codes of all the words present in the name of the stored files (section 4.6.1). Upon receiving the Query message, a super-probe must first break up the words contained in the message's search criterion on any non-alphanumeric characters (anything but letters and numbers); a space is the standard separator between words but the characters: "_" and "-" may also be used. Remember that the message's search criterion may be the name of the file to be searched. After that, the super-probe must determine, for each probe, if the hash code (appendix A.3) of each obtained word (lower case) is present in the list of hash codes associated with the probe. Only when a match is found for all obtained words will the Query message be forwarded to the probe.

A node receiving a Query message should only respond with a Query-Hit message if it supports a common security mode with the requesting client. The security modes supported by the client are indicated in the Query message. The nodes should have this behavior because the client will only be able to connect to the node and retrieve the results of a configured test session if both elements support a common security mode (section 4.2.4). Moreover, a super-probe should only forward a Query message to a probe if the probe supports a common security mode with the requesting client. The super-probe has the information of the security modes supported by each probe connected to it (section 4.3.5.2).

A probe must reset the TCP connection to its super-probe and try to connect to another super-probe of its measurement group in case it receives a Query message it cannot process. This is because the super-probe should not have sent this message in the first place.

In case a node is supposed to process a Query message, it must verify if it is sharing files that satisfy the message's search criterion. Both probes and super-probes process this criterion in the same way, as described next:

i. If the search criterion has the name of the file to be searched, the node must verify if it is sharing that file. In this search, the file name extension should be ignored. Note that the measurement process may have produced a file with extension ".res" in case there were no errors during the measurements or a file with extension ".err" otherwise (section 4.4.3). Files with extension ".res" are stored in the results directory and files with extension ".err" in the error directory. Ignoring the extension will allow the search process to discover both ".res" and ".err" files. The node will send to the client the complete file name, and the client, in case of an error file, may decide to download the file to obtain additional information on the type of error that occurred;

ii. If the search criterion is a set of keywords, it is recommended that the node break up the words on any non-alphanumeric characters (anything but letters and numbers). Only the files which names match all the keywords should be included in the Query-Hit message. All Files stored in the results and error directories should be analyzed.

If a node is sharing files that satisfy the search criterion of the received Query message, the node must send a Query-Hit message to the requesting client. The Query-Hit messages must only be sent along the same path that carried the incoming Query message, as described in the section 4.2.3.

In Figure 28 is given an example of a Heavy Data File search.

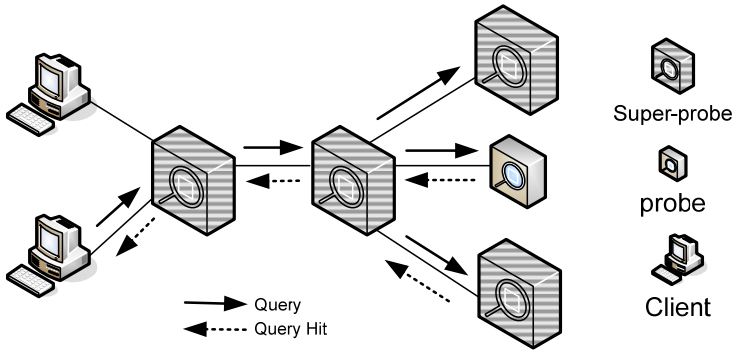

Figure 28. Data query

After sending a Query message to its super-probe, a client should wait for the first Query-Hit message to be received for a given timeout. The client must discard any Query-Hit messages, sent in response to the Query message, if the first Query-Hit message is not received before the timeout occurs. To do so the client must remove the record corresponding to the Query message from its CLM (section 4.2). However, if the first Query-Hit message is received before the configured timeout, all subsequent Query-Hits should be considered until the user looses interest in this information.

After receiving the first Query-Hit message, the client shows to the user the set of files that matches the search criterion. Upon receiving subsequent Query-Hit messages, additional files that match the search criterion and additional nodes that may be sharing the files may be identified, and this information is also updated to the user. With this information the user can choose which files he wants to download. Also if the file is available from multiple sources, the client may use the multi-source download feature to speed up the download process. The file download is accomplished outside the DTMS-P2P network, as described in section 4.7.

4.7 Data download

There are many situations where there is transference of data (data download) between two elements of the network. This transference occurs, for example, in the file replication process (section 4.5) and in the results retrieval process (section 4.6). But there are more situations where there is transference of data between the elements of the DTMS-P2P network and they will be described later (sections 4.8.3 and 4.9.4.2).

In the DTMS-P2P system the data download is performed directly between the requesting element and the node where the file is stored using HTTP, outside the DTMS-P2P network. A direct TCP connection between the two involved elements must be established and the file transference should be performed through the established connection. To open a connection to a remote node an element should proceed as described in section 4.3.5.1.

Moreover, the data download can be made from multiple sources when the file is stored in multiple locations.

Also, each file download process must be processed in a different thread because the elements involved in the file transfer process should not block while uploading/downloading a file.

The overall file transfer process will be described in the next sections.

4.7.1 Data download between two elements

Before starting a file download process the requesting node should reserve the required space to store the file to be downloaded. Having this behavior the node can guarantee that this storage space will be available to store the required file. To reserve it, the node should create a temporary file with the same size of the file to be downloaded. Then the content of the required file can be stored into the temporary file. After the end of the file download process, the node should rename the temporary file to the name of the downloaded file.

As described before, the DTMS-P2P system uses HTTP as the file download protocol. Implementations should support HTTP 1.1 [RFC2616]. The full specification of this version of the protocol is available in [RFC2616]. In this description we will only focus on the basic issues.

After being connected to the source element, the request to download a file is initiated by sending an HTTP GET message with the following contents:

GET /get/<File Name> HTTP/1.1<cr><lf>

User-Agent: DTMS-P2P client<cr><lf>

Host: x.y.z.n:p<cr><lf>

Connection: Keep-Alive<cr><lf>

Range: bytes=0-<cr><lf>

<cr><lf>

where <File Name> is the name of the file the requesting element wants to download. For example, if the requesting element wants to download the file which name is 00000000000000000000000000000000_192.168.0.4.21164_00000000000000000000000000000000_192.168.0.1.22368_1175895448445_ping 192.168.0.2 -n 200.res and is stored at the source node with IP address 192.168.0.1:22368, a download request would be initiated as follows:

GET /get/00000000000000000000000000000000_192.168.0.4.21164_00000000000000000000000000000000_192.168.0.1.22368_1175895448445_ping 192.168.0.2 -n 200.res HTTP/1.1<cr><lf>

User-Agent: DTMS-P2P client<cr><lf>

Host: 192.168.0.1:22368<cr><lf>

Connection: Keep-Alive<cr><lf>

Range: bytes=0-<cr><lf>

<cr><lf>

The User-Agent string may be any name chosen by the developer. For example, it may be assumed that the User-Agent header should contain the name of the DTMS-P2P element sending the message (e.g. *DTMS-P2P probe*, *DTMS-P2P super-probe* or *DTMS-P2P client*).

The Host header is required in HTTP 1.1 and specifies the address of the remote node (in this case the source node). It is usually not used by the receiving node, but its presence is required by the protocol.

The Connection header tells the remote host if the connection should be closed when the transfer is finished or not. “Connection: close” means that the connection must be closed after the transfer. “Connection: Keep-Alive” or no Connection header means the connection must be kept open. The requesting element may then issue another request for another range of a file or another file. The request may be sent before the previous transfer is finished. This is called a Persistent Connection, and it is described in section 8.1 of RFC 2616.

In the DTMS-P2P system, if the requesting node demands a source node to keep the connection active (Keep-Alive), the node should only keep it active for a configured timeout (say 24 hours). If no messages are received within the timeout period, the connection should be closed.

The Range header is used to request the complete file content when defined as “bytes=0-“ or the range of bytes to be downloaded, for example, “bytes=766-10083”. Note that the Range header does not have to specify both start and end positions.

The node receiving this download request should verify if it is sharing the required file. If not, the node should verify if a file with the same name but an extension “.err” is stored. This file describes the errors that may have occurred during the measurement process (section 4.4.3). In this case, the “.err” file should be downloaded in place of the “.res” one.

In case the responding node is sharing the required file or the corresponding error file, it should respond with HTTP 1.1 compliant headers such as:

HTTP/1.1 200 OK<cr><lf>

Server: DTMS-P2P probe<cr><lf>

Content-Type: result/res<cr><lf>

Content-Length: 11278<cr><lf>

<cr><lf>

With this response, the source node accepts the received request. Afterwards, the file is sent and should be read up to, and including, the number of bytes specified in the Content-Length provided in the node's HTTP response message.

In the HTTP response message, the Server header, as the User-Agent header, may contain the name of the node sending the message. For example it may be *DTMS-P2P probe* or *DTMS-P2P super-probe*.

The Content-Type header should be used to specify the type of the file to be downloaded. For example, in case the file contains the results of a test session, the Content-Type header should be equal to "result/res". If the requested file contains the description of the error that occurred during a measurement process, the Content-Type header should be equal to "error/err".

The Content-Length header is used to specify the length in bytes of the content of the file to be downloaded.

However, in case the remote node verifies that is not sharing the requested file it should reply with a file not found message:

HTTP/1.1 206<cr><lf>

<cr><lf>

In this case, both elements should close the established connection.

When the GET message requests for a range of the file to be downloaded, e.g., with Range: bytes=766-10083<cr><lf>, the response is a Partial Content message:

HTTP/1.1 206 Partial Content<cr><lf>

Server: DTMS-P2P probe<cr><lf>

Content-Type: result/res<cr><lf>

Content-Length: 9318<cr><lf>

Content-Range: bytes 766-10083/11278<cr><lf>

<cr><lf>

The Content-Range header is used to specify the range of bytes to be downloaded and the file size in bytes.

Implementations should allow the node's administrator to configure the maximum number of file transfers a node should handle at the same time. This number is represented by a number of slots. The start of a file transfer (download or upload) requires a free slot. If all slots are taken, the node is said to be busy. If this is the case, the node should answer with the following message when receiving a download request:

HTTP/1.1 503<cr><lf>

<cr><lf>

When a storing node is busy, if the requesting node demanded it to keep the connection active (Keep-Alive), the requesting node should periodically (30 seconds by default) request the file download again until the remote node is able to upload the file.

In addition, if the requesting element is a node (probe or super-probe), before requesting the download of a given file, it should also verify if it has a free slot to download the file. If all slots are taken, the node should wait until it gets a free one to process the request and download the required file.

During the download process, headers unknown to the DTMS-P2P elements must be ignored.

The HTTP messages used during the file transfer process must be transmitted in ASCII format, using the ISO-8859-1 character encoding variant.

DTMS-P2P elements should not attempt to download multiple files from the same source at once. Files should be locally queued instead.

The implementation should allow the local administrator to configure the maximum speed at which a file should be downloaded by a remote node.

After the end of a successful download or upload process, a node should use the average transfer rate of the file transfer to determine the highest average transfer rate of the last 10 downloads or uploads, respectively. The node's download and upload speed should be consider to be equal to the highest average transfer rate of the last 10 downloads or uploads, respectively.

4.7.2 Firewalled nodes

Before starting the file transfer process, the involved nodes should establish a connection to each other to be used during the download process. When the storing node (source node) is not firewalled, the requesting node (node where the file will be replicated) can directly open a connection to it (section 4.3.5). However, the storing node may be behind a firewall or may not accept incoming connection requests. In these cases, the download process is only possible if the requesting node can accept incoming connection requests (file upload). Therefore, because a firewalled element cannot accept incoming connection requests, the

file transference process is only possible when one of the involved elements is not firewalled.

When the storing node is behind a firewall or for any other reason cannot accept incoming connection requests, the requesting element must send a Push request message (appendix A.2.29) to the storing node through the DTMS-P2P network, asking it to establish the connection for data download. After sending this message, the requesting node should wait a configured timeout for the connection establishment. If the connection is not opened during this timeout, the requesting element should stop the download process. The Push message should be forwarded to the storing node as described in section 4.2.1.

After receiving a Push message, a super-probe should first verify if it is directly connected to the destination node. If so, it verifies also if the destination node and the requesting node support a common security mode. If not, the super-probe discards the message since the destination node and the requesting node will not be able to establish a connection for data download (section 4.2.4). Otherwise, the message must be forwarded to the destination node.

After receiving a Push message, the storing node must first verify if it can accept the request for file upload. If the node is not sharing the required file (results file or error file) or the shared file does not have the same size as requested in the Push message, the node should not accept the request. Otherwise, if the node is sharing the file, it must open a connection to the requesting element (section 4.3.5) and send it a HTTP GIV message. This message prompts the remote node to download the required file using the established connection. This message must be transmitted in ASCII format, using the ISO-8859-1 character encoding variant, in the following form:

GIV <file name>:<file size><cr><lf>

User-Agent: <the name of the sending node><cr><lf>

Server: <IP Addr:port><cr><lf>

<cr><lf>

where <file name> is the name of the file to be downloaded and <file size> is its size in bytes. The User-Agent field may have the name of the node sending the message. For example it may be *DTMS-P2P probe* or *DTMS-P2P super-probe*. The Server field must have the IP address and port number where the node sending the message is running.

After sending the HTTP GIV message, the storing node must wait to receive an HTTP GET request message from the requesting element, during a configured timeout. The GET request and the remaining file download process is identical to the one described in section 4.7.1.

The requesting element should ignore the <file name> field in the received HTTP GIV message, and request the file it wants to download. The storing node must allow the requesting element to request any file, and not just the one specified in the Push message. Next we give one example of an HTTP GIV message.

GIV
00000000000000000000000000000000_192.168.0.4.21164_000000000000000000000000000000_192.168.0.1.22368_1175895448445_ping 192.168.0.2 -n 200.res:11278<cr><lf>

User-Agent: DTMS-P2P probe<cr><lf>

Server: 192.168.0.1:22368<cr><lf>

<cr><lf>

If the TCP connection is lost during a Push initiated file transfer, it is strongly recommended that the storing node attempts to re-connect. This is important, since the requesting element may not be able to send another Push message to the storing node.

To be able to download files from firewalled nodes, a client must be able to accept incoming connection requests. Thus, the client should not be firewalled and should be listening for incoming requests in a given IP address and port number. But the client element should only accept an incoming connection request if it is to be used to download a file from a remote firewalled node. The client element should never accept incoming connection requests from elements trying to connect to the network.

4.7.3 Multi-source download

When the file to be downloaded is stored in more than one source, the requesting node should retrieve the file from multiple locations. The multi-source download may be advantageous because the requesting element can download different sections of the file at the same time from different locations. This behavior may decrease the time an element will need to download a given file. There will be an advantage when the download speed from multiple sources is higher than the download speed from the fastest individual source.

The node should at least download a given portion of the content of the file (minimum download size – minDownloadSize), say 1 MB = 1048576 bytes, from each remote node. Using the minDownloadSize value, the node may determine the maximum number of node sources (maxNumberOfSources) it can use to download the overall content of the file. This value can be computed by dividing the file size by the minDownloadSize. If this value is greater than the number of available sources (numAvailableSources), the node should use all the available sources. Otherwise, the node should only use the required maxNumberOfSources. In this last case, the node should sort the list of sources using their upload speed. The faster sources should be preferred. If an error occurs while downloading from a given source, one of the remaining free sources should be used to continue the file download.

After computing the number of sources to be used (numOfSourcesToBeUsed), the requesting node should divide the total file length in sections of equal size. The number of sections should be equal to the numOfSourcesToBeUsed. Each section should be downloaded from a different download source in parallel. Because the numAvailableSources may be less than the maxNumberOfSources required to download

the minDownloadSize from each source, the length of a section may be greater than the minDownloadSize.

Whenever a section is completely downloaded from a given source or the requesting element gets a new download source, the requesting element should use the new free source to improve the file download process. A requesting element may receive the information of a new download source, for example, if a new Query-Hit message is received during a search process (section 4.6.2). In both cases, the element should use the new free source because, if the file is downloaded in parallel from all the available sources, the download process may be faster.

To be able to use a new free download source, an element should identify the slower download source currently being used. If this source has to download yet more than, say the minDownloadSize, the remaining bytes of the section to be downloaded may be divided between this source and the free one (a process called split/steal). Then, portions of the remaining bytes can be simultaneously downloaded from both sources.

In this situation, it is very likely that one of the download sources to be used is faster than the other one. Thus, to improve the download process, the faster download source should be used to download a larger portion of the remaining bytes. To determine the number of bytes to be downloaded from one source and from the other one, we should consider that the time required to download the two parts of the remaining bytes from both sources should be equal. Using this, we prevent the faster download source to download the required number of bytes in less time, which would lead to the overall process to be repeated again.

Let the speed and number of bytes to be downloaded from source i be denoted by v_i and b_i respectively, and let the time required to download b_i bytes from source i be denoted by T_i. Then, the number of bytes to be downloaded from the faster and slower sources is given by

$$\begin{cases} T_1 = T_2 \\ b = b_1 + b_2 \end{cases} \Leftrightarrow \begin{cases} \frac{b_1}{v_1} = \frac{b_2}{v_2} \\ b = b_1 + b_2 \end{cases} \Leftrightarrow \begin{cases} b_1 = \frac{\frac{v_1}{v_2} b}{1 + \frac{v_1}{v_2}} \\ b_2 = \frac{b}{\frac{v_1}{v_2} + 1} \end{cases}$$

The download speed v_i is set initially to the value declared by the source but is progressively corrected based on measurements carried out by the requesting element as it downloads the data.

After computing the number of bytes to be downloaded from each source, the element should use the free download source to download the last bytes of the divided section. The source currently in use should be used to continue receiving the first bytes of the divided section. Thus, the process currently downloading the section must continue downloading the required portion of the remaining bytes and not all the section as it was initially

scheduled. When this required number of bytes is downloaded, this process should close the connection to the storing node to not receive more data from it. The storing node should also stop sending data and close the connection because the requesting element will not use this data anymore since it is being downloaded from another source. Doing this, it is guaranteed that the remaining bytes of the divided section are downloaded from the two remote sources in parallel.

The requesting element should not use a new free source when (i) the number of bytes that remain to be downloaded is lower than minDownloadSize or (ii) the source is slower than the source being tested and the number of bytes it should download is smaller than minDownloadSize. This is to avoid spending the overhead of the splitting process when the data to be downloaded is small.

In this case, the new free source should be tested against all other sources, by increasing order of download speed, until a source that can split data download with it is found or the fastest source has been tested. Although these sources provide faster downloads they may be downloading larger portions of data which may bring the splitting process to an advantage. Using this process, it is guaranteed that, whenever possible, the requesting element is always downloading different sections of the file and from different sources at the same time, which improves the overall download process.

When the element finishes downloading a given section of the file from a remote source, it should verify if the section was or not successfully downloaded. In case an error occurred during the section download and no bytes were downloaded from the remote source, the element should not use the source any more. If no more sources are available to download the required file, the element should stop the download process and remove the temporary file from its storage space.

While downloading from multiple sources the requesting element should keep all connections opened until the file is completely downloaded, since a source can used to download several parts of the file. Also, the source node should relay to the requesting element the task of closing connections.

For each download section the transfer process should be performed as described in section 4.7.1. The element should store each file section at the correct position of the temporary file used to store the content of the file. When all the sections of the file are downloaded the node should rename the temporary file to the original file name.

4.7.4 Encryption

After the connection set-up process and if the elements chose to use the authenticated or the encrypted security modes, all HTTP control messages (GET, GIV, etc) must be encrypted using AES and authenticated using HMAC as described in section 4.1. However, the data messages (used to transfer the content of a file during file download) are never authenticated but are encrypted in the encrypted mode. Authenticating data messages would increase significantly the load of the download process when downloading large files.

4.8 Light Data

The Light Data Files (LDFs) are one of the types of measured data files used in the DTMS-P2P protocol. A LDF stores the round-trip time (RTT) statistics between a super-probe and all the elements connected to it (probes, super-probes and clients of its measurement group and super-probes of other measurement groups). The information stored in a LDF is called light data. Any super-probe will store the most recent LDFs of all super-probes (including its own). The complete set of LDFs is called the Compiled Light Data File (CLDF). Each super-probe must periodically build its LDF and broadcast it to all other super-probes of the network.

A node's administrator must configure the parameters that the node must use to periodically generate its LDF, when operating in super-probe mode. These parameters are: (i) the interval time between LDFs generation (lightDataUpdateInterval – 30 minutes by default), (ii) the number of echo requests to be send for RTT calculation (4 by default) and (iii) the interval time between these echo requests (1 second by default).

At any time, a user may use a client element to obtain a CLDF directly from the super-probe to which the client is connected to or from any other super-probe of the network.

Because CLDFs only contain the RTT statistics between super-probes and the elements connected to them, they provide a coarse, but updated and fully available, view of the network status.

4.8.1 Light Data file format

We have adopted XML as the LDF and CLDF formats because of its readability which facilitates the exchange of these files between the elements of different implementations of the DTMS-P2P protocol. Both files use exactly the same formats with the Document Type Definition (DTD) represented in Figure 29.

```
<!DOCTYPE LightData [
  <!ELEMENT LightData (SuperProbe*)>
  <!ELEMENT SuperProbe (DTMS_P2P_Element+)>
  <!ELEMENT DTMS_P2P_Element (address,mode,date,rtt)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT mode (#PCDATA)>
  <!ELEMENT date (#PCDATA)>
  <!ELEMENT rtt (min,avg,max)>
  <!ELEMENT min (#PCDATA)>
  <!ELEMENT avg (#PCDATA)>
  <!ELEMENT max (#PCDATA)>
  <!ATTLIST SuperProbe address CDATA #REQUIRED>
  <!ATTLIST SuperProbe groupID CDATA #REQUIRED>
  <!ATTLIST SuperProbe lightDataUpdateInterval CDATA #REQUIRED>
  <!ATTLIST SuperProbe numOfEchoRequests CDATA #REQUIRED>
  <!ATTLIST SuperProbe intervalBetweenEchoRequests CDATA #REQUIRED>
  <!ATTLIST DTMS_P2P_Element id ID #REQUIRED>
]>
```

Figure 29. Light Data File XML DTD.

In this XML DTD the "SuperProbe" element comprises the light data of a given super-probe. A LDF comprises only one "SuperProbe" element. A CLDF may contain different

"SuperProbe" elements, each one related to a given super-probe. Each "SuperProbe" element has different "DTMS_P2P_Element" child elements. These last ones contain their own child elements, which are: "address", "mode", "date" and "rtt". The "address" element represents the address of the remote DTMS-P2P element connected to the super-probe generating the respective light data. The "mode" element should be the code of the mode of the element: 0 – probe, 1 – super-probe of the measurement group of the generating super-probe, 2 – client and 3 – super-probe of another measurement group. The "date" element should be equal to the difference, measured in milliseconds, between the time the RTT calculation was started and midnight, January 1, 1970 UTC. The "rtt" element comprises the computed RTT statistics to the respective element. These statistics are represented by the child elements "min", "avg" and "max" which represents the obtained minimum, average and maximum RTT statistics, respectively. The RTT statistics ("rtt" child elements) should be defined in milliseconds. If the generating node is not able to compute the RTTs to a given DTMS-P2P element the "rtt" child elements must be set to "-1". This situation may happen if the remote element is firewalled or for any other reason does not respond to ICMP echo requests. The "SuperProbe" element has some attributes. The "address" attribute represents the address of the generating super-probe, and the "groupID" attribute is the Group ID of the super-probe's measurement group. The "lightDataUpdateInterval" and "intervalBetweenEchoRequests" attributes should be defined in milliseconds. They represent the interval time the super-probe is configured to wait between successive light data generation and the interval time between the echo requests sent to compute the RTTs, respectively. The "numOfEchoRequests" attribute is the number of echo requests the generating super-probe is configured to send, when computing RTTs.

Considering the network of Figure 30, an example of a LDF produced by the super-probe 192.168.1.1 is depicted in Figure 31. An example of a CLDF obtained from the same super-probe is depicted in Figure 32.

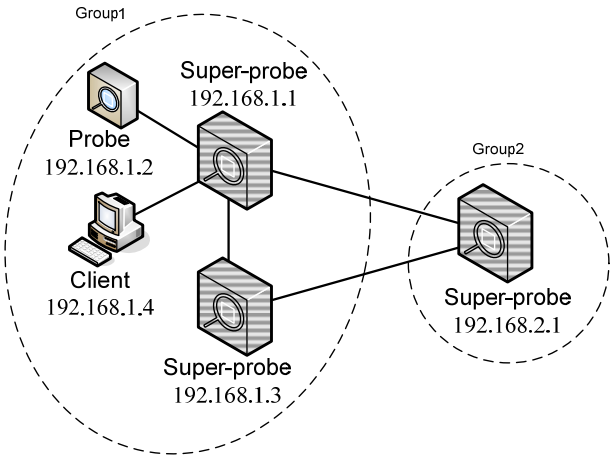

Figure 30. Light Data.

```
<?xml version="1.0"?>
<LightData>

    <SuperProbe address="192.168.1.1:22368"
groupID="00000000000000000000000000000001" lightDataUpdateInterval="60000"
numOfEchoRequests="4" intervalBetweenEchoRequests="1000">

      <DTMS_P2P_Element id="n0">
        <address>192.168.1.2:22368</address>
        <mode>0</mode>
        <date>1175850535348</date>
        <rtt>
          <min>0.0</min>
          <avg>0.0</avg>
          <max>0.0</max>
        </rtt>
      </DTMS_P2P_Element>

      <DTMS_P2P_Element id="n1">
        <address>192.168.1.3:22368</address>
        <mode>1</mode>
        <date>1175850538405</date>
        <rtt>
          <min>0.0</min>
          <avg>0.0</avg>
          <max>0.0</max>
        </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n2">
      <address>192.168.1.4:22368</address>
      <mode>2</mode>
      <date>1175850541449</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n3">
      <address>192.168.2.1:22368</address>
      <mode>3</mode>
      <date>1175850544502</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

  </SuperProbe>

</LightData>
```

Figure 31. Light Data File example.

```
<?xml version="1.0"?>
<LightData>

  <SuperProbe address="192.168.1.1:22368"
groupID="00000000000000000000000000000001" lightDataUpdateInterval="60000"
numOfEchoRequests="4" intervalBetweenEchoRequests="1000">

    <DTMS_P2P_Element id="n0">
      <address>192.168.1.2:22368</address>
      <mode>0</mode>
      <date>1175850835356</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n1">
      <address>192.168.1.3:22368</address>
      <mode>1</mode>
      <date>1175850838396</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n2">
      <address>192.168.1.4:22368</address>
      <mode>2</mode>
      <date>1175850841450</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n3">
      <address>192.168.2.1:22368</address>
      <mode>3</mode>
      <date>1175850844499</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

  </SuperProbe>

  <SuperProbe address="192.168.1.3:22368"
groupID="00000000000000000000000000000001" lightDataUpdateInterval="60000"
numOfEchoRequests="4" intervalBetweenEchoRequests="1000">

    <DTMS_P2P_Element id="n0">
      <address>192.168.1.1:22368</address>
      <mode>1</mode>
      <date>1175850840268</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n1">
      <address>192.168.2.1:22368</address>
      <mode>3</mode>
      <date>1175850843317</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

  </SuperProbe>

  <SuperProbe address="192.168.2.1:22368"
groupID="00000000000000000000000000000002" lightDataUpdateInterval="60000"
numOfEchoRequests="4" intervalBetweenEchoRequests="1000">

    <DTMS_P2P_Element id="n0">
      <address>192.168.1.1:22368</address>
      <mode>3</mode>
      <date>1175850857106</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>

    <DTMS_P2P_Element id="n1">
      <address>192.168.1.3:22368</address>
      <mode>3</mode>
      <date>1175850860141</date>
      <rtt>
        <min>0.0</min>
        <avg>0.0</avg>
        <max>0.0</max>
      </rtt>
    </DTMS_P2P_Element>
  </SuperProbe>

</LightData>
```

Figure 32. Compiled Light Data File example.

4.8.2 Light Data generation and distribution

After starting a node in super-probe mode or if the node is promoted to a super-probe, it should periodically (lightDataUpdateInterval) build a new LDF by computing the RTTs to each element it is connected to.

A new LDF will replace an older one. A new LDF should be broadcasted to all other super-probes over the DTMS-P2P network using a Light Data message (appendix A.2.34). This process could also be performed using HTTP between each pair of super-probes. However, due to network constraints regarding firewalls, not all super-probes may be able to directly connect to a generating super-probe and request its LDF using HTTP. Moreover, due to the limit constraints of a super-probe's LAC and CKN, it is not possible to guarantee that a super-probe knows all other super-probes of the network. Flooding the LDF using the DTMS-P2P overlay network is a faster process and it guarantees that a LDF of a generating super-probe is received by all super-probes at a maximum distance of TTL hops from it.

The content of a LDF may be larger than the payload of a Light Data message (messages should not be larger than 4 KB – appendix A.2.1). In this case, the LDF must be fragmented. As all messages, to support the fragmentation process each Light Data message includes a Message ID, equal for all fragments of the same LDF, and uses the Cont. Frag. Index (Content Fragment Index) and M (More Fragments) fields of the message header (appendix A.2.1). Fragments should be of maximum size, whenever possible.

When a super-probe is connected to more than one super-probe, it first sends one fragment to all super-probes, and only after this starts sending the next fragment. Another alternative would be to send all fragments to one super-probe, and only after this start sending to the next-super-probe.

When a Light Data message is received for the first time by a super-probe and the message includes a full LDF, the LDF is stored at the super-probe, replacing an older one if it exists (with a different Message ID). The LDF is stored with the name *IP Address.Port_MessageID.xml* where *IP Address* is the IP address of the generating super-probe, *Port* is the port number where the super-probe is listening for incoming connection requests and *MessageID* is the Message ID of the Light Data message that transported the LDF. The Message ID should be used to control the flooding process in support to the one described in section 4.2: if a Light Data message with a Message ID already present in a file name is received, then the message must be discarded; otherwise, it is the first time that the message is received by the super-probe, and the super-probe must flood the message, after storing the corresponding LDF.

The case of fragments is handled in a different way. The first fragment of a LDF arriving at super-probe will force the creation of a temporary folder, where all other fragments (with the same Message ID) will be stored, with name *IP Address.Port_MessageID*. Each fragment will be stored in a temporary file with name *IP Address.Port_FragmentIndex.tmp*, where *FragmentIndex* is the fragment number (message's header Cont. Frag. Index field - appendix A.2.1). When all fragments have been received, the various fragment files are merged into a file with name *IP*

Address.Port_MessageID.xml, replacing an older one if it exists, and the temporary folder is destroyed. If at least one fragment is not received within a time interval, counted from the reception of the first received fragment, then the super-probe must destroy all temporary files and the temporary folder. This timeout may be equal to, say, number of fragments times the configured timeout to receive a message from a remote node. In support to the flooding mechanism described in section 4.2, the flooding of fragments is controlled in the following way: if a Light Data message with a Message ID present in a file name is received, then the message must be discarded (this means that the LDF was already reassembled at the super-probe); else if the Message ID is present in a temporary folder name and the fragment number is present in one of the file names of this folder, then the message must also be discarded; otherwise, the super-probe must flood the message, after storing the corresponding LDF fragment.

In both cases, a message to be flooded is sent to all (super-probe) neighbors except the super-probe that sent the message and the super-probe that generated the message (if directly connected to the super-probe).

4.8.3 Light Data retrieval

As mentioned before, any client connected to any super-probe can retrieve the information stored in a Compiled Light Data File (CLDF) directly from its super-probe or from any other super-probe of the network. Due to the fact that the LDFs of a super-probe are periodically updated, a super-probe should only produce the CLDF when a client requests to download it.

The CLDF is requested using HTTP. To request the file, the client opens a new TCP connection to the required super-probe (section 4.3.5.1) and requests the download of the *LightData.xml* file, which contains the CLDF. This request is performed as described in section 4.7.1. Since a client does not know the size of the remote super-probe's CLDF, it must request the download of the complete *LightData.xml* file content; to do so, the client uses the HTTP GET request message with a Range header defined as "bytes=0-".

When a super-probe receives a request for downloading the *LightData.xml* file, it starts by building the CLDF with the information of its own LDF and of the LDFs received from other super-probes. This file is built using the XML DTD presented in section 4.8.1. After building the CLDF, the super-probe responds with an HTTP response message accepting the download request. The HTTP response message's Content-Type header must be equal to "lightData/xml". Using the Content-Length header of the received HTTP response message the client can determine the length of the super-probe's CLDF. After completing the CLDF upload to the requesting client, the super-probe must remove the *LightData.xml* file from its storage space. However, because a super-probe may receive simultaneous requests for downloading the Light Data File, implementations must control the creation and deletion of the super-probe's CLDFs.

A node receiving a *LightData.xml* file download request must only accept to process the request if it is a super-probe. If it is not a super-probe, it should reject the request and close the connection to the requesting client.

Usually a client should be used to obtain the CLDF directly from the super-probe to which it is connected, but a user may also configure a client to obtain the file from any other super-probe of the network. To do so the client can be configured to obtain the list of nodes of a given measurement group, using the mechanism described in section 4.4.2.2.2, and the super-probe selected among the nodes of the required measurement group. The user can also enter manually the address of a super-probe from where he wants the CLDF to be downloaded.

In case the selected super-probe is not the super-probe to which the client is directly connected, it may be possible that the client cannot open a direct connection to the selected super-probe to request the HTTP CLDF download. This situation may happen if the selected super-probe is behind a firewall or for any other reason cannot accept connection requests from the client. When trying to request the CLDF from the selected super-probe, the client must first try a direct connection; in case an answer is not received within a configured timeout, the client must send a Push message (appendix A.2.29) to the selected super-probe, using the DTMS-P2P network. This message directs the selected super-probe to open a connection to the requesting client in order to upload the *LightData.xml* file. This download is performed in a similar way as the one described in section 4.7.2. In this case and because the client does not know the size of the remote's super-probe CLDF, the file size field of the Push message must be set to zero. In case the receiving node is not a super-probe it must not process the request: this node may be a super-probe that was demoted to a probe.

After completely downloading the super-probe's CLDF, the client must store it in its download directory with a different name. The name of the CLDF may be "*IP Address.Port_LightData_<time>.xml*", where IP Address and Port represent the IP address and port number of the remote super-probe and <time> field should be equal to the difference, measured in milliseconds, between the time the CLDF was downloaded and midnight, January 1, 1970 UTC. Remember that a client may be used to download the CLDFs of different super-probes and may be used to request the download of the CLDF of a given super-probe several times.

4.8.4 Light Data clean interval

After a certain interval a super-probe may be storing LDFs of super-probes that are no longer connected to the network or that were demoted to probes. These LDFs should be deleted, as soon as possible, from the storage space of the super-probe. However, it is not always possible for a super-probe to verify if a remote super-probe is not a super-probe anymore or if it was disconnected from the network. Moreover, even knowing the light data update interval of a remote super-probe, it is difficult to predict when a new LDF should be received from it, because the construction of LDFs takes a variable amount of time that depends strongly on the number of nodes it is connected to. In order to circumvent this problem, the super-probe's administrator should define an interval time, called light data clean interval (default 24 hours), to delete the LDFs of all super-probes that are not directly connected to it. This allows for periodically removing LDFs from super-probes that no longer exist.

4.9 Additional client actions

4.9.1 File action request

A user can request the replication and/or deletion of Heavy Data Files that were generated by measurements he configured. To do so the user client sends a File Action Request message (appendix A.2.30), that contains the client IP address and measurement group, the name of the Heavy Data File and the type of action (replicate and/or delete). The destination node (probe or super-probe) upon receiving this message performs the requested action and afterwards sends a File Action Response message (appendix A.2.31) to inform the client. This is illustrated in (Figure 33).

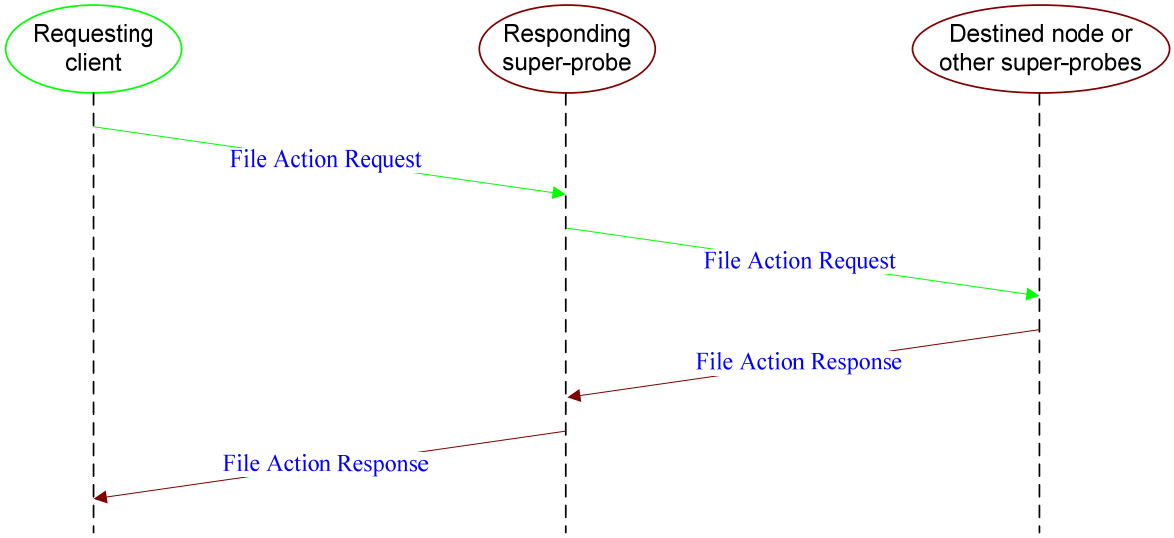

Figure 33. Messages exchanged during the file action request process.

A user should only request the replication and/or deletion of a Heavy Data File through the same client used to configure the corresponding measurements. This should be done to prevent users to delete the files produced by test sessions configured by other users. To do so, the client checks if its IP address and measurement group match the corresponding information contained in the name of the Heavy Data File, as inserted by the user (section 4.4.2.6.2).

This verification is also performed by all nodes traversed by the File Action Request message, which verifies if, inside the message, the client IP address and measurement group match the corresponding information contained in the name of the Heavy Data File. If not, the message should be discarded and, if the message was received from a super-probe, the receiving node must close the connection with it, since super-probes must not flood invalid messages. In case of rejection from the super-probe directly connected to the client, the super-probe must respond with a File Action Response message to inform the client that the request was not accepted.

In addition, the super-probe that is directly connected to the client verifies if the client IP address contained in the message matches the IP address read from the socket connection with the client. It also verifies if the client measurement group contained in the message equals its own. If one or both verifications fail, the super-probe must close the connection

to the client. This verification must be performed to prevent a user to try to hide the identification of the client he is using, to be able to request the replication and/or the deletion of files produced in test measurements not configured using the respective client.

When the File Action Request message reaches the destination node, it must verify if it is sharing the required file. If not, the node must send a File Action Response message back to the requesting client informing it that the request was rejected.

If the remote client is requesting the node to replicate the file, the node should replicate it as described in section 4.5 to produce the required number of replicas. The number of replicas is indicated in the File Action Request message. If the node is already in the process of replicating the file, the requested actions (replication and/or deletion) must be rejected.

If the remote client requested replication and deletion, the file should not be deleted in case the node was not able to perform all the required replications.

If the node is a probe, after deleting the file from its storage space, it verifies for all words present in the deleted file name, if there is no other file being shared with that word in the file name. If yes, the probe should send a List of Shared Files message (appendix A.2.8) to its super-probe with the hash codes of the words that are no longer present in any of the files it is sharing. The super-probe must remove these words from the probeListOfSharedFiles related to the sending probe (section 4.6.1). In this way, the super-probe maintains updated information regarding the files being shared by its probes.

4.9.2 Resources request

Whenever a user wants to known the available resources at a given node of the network he should use a client to send it a Resources Request message (appendix A.2.32). This message should be forwarded to the destination node as described in section 4.2.1. The Resources Request message may be used to get the available resources at a given super-probe and all probes under its control.

After receiving a Resources Request message, the receiving node should respond with a Resources message (appendix A.2.33) with information about the node's available resources (Figure 34). This message should be routed back to the requesting client as described in section 4.2.1.

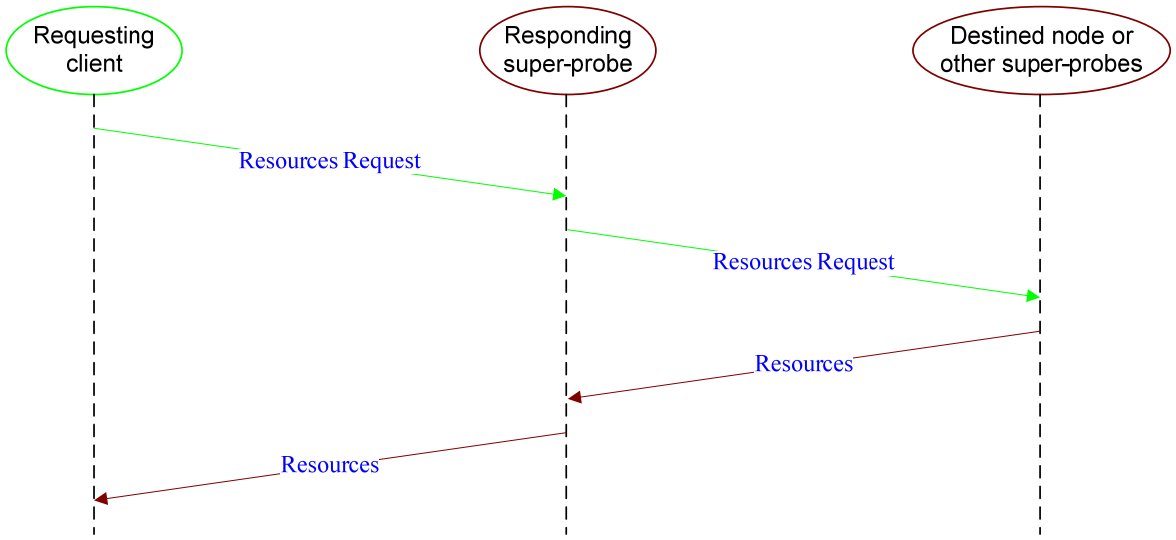

Figure 34. Messages exchanged during a resources request process.

When the user prompts the client to obtain information about the available resources at a super-probe and all its probes, the Resources Request should be forwarded to each probe in turns when it arrives at the super-probe. After sending the Resources Request message to a probe, the super-probe must not send to the probe any more messages until it receives a Resources message from it. The super-probe must only wait to receive the Resources message for a configured timeout. In case the message is not received within this timeout, the super-probe must close the connection to the remote probe. While waiting to receive the Resources message, a super-probe must process any other message received from the probe or from other elements connected to it. Remember that a probe may send some messages to its super-probe whenever a change occurs (e.g. List of Shared Files (sections 4.4.3, 4.6.1 and 4.9.1), List of Supported Monitoring Modules (section 4.4.1.2), etc). A super-probe only sends a Resources Request Message to a probe after receiving the Resources message from the previous one or a timeout has occurred. The Resources Message sent by the super-probe carries information about the super-probe and all probes that have answered to the request. This process is illustrated in Figure 35.

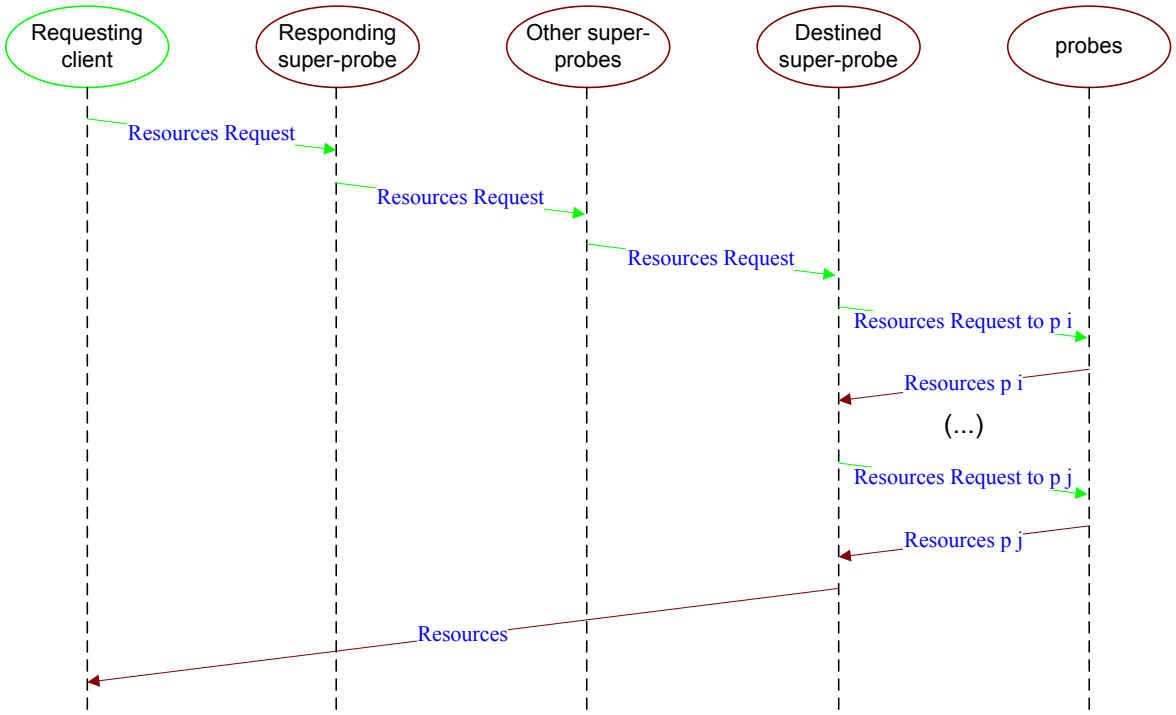

Figure 35. Messages exchanged during a resources request process (sp and p).

4.9.3 Connect to node request

A user may want to actively request a given node of the network to connect to another node. This possibility is advantageous because the user will be able to interconnect two measurement groups which are not interconnected, possibly because the nodes of both measurement groups do not know each other; also he will be able to integrate to the network a node not connect to it.

To request a node (the first node) of the network to connect to another node (the second node), a client must send a Connect to Node Request message (appendix A.2.35) to it. This message should be forwarded to the destination node as described in section 4.2.1.

After receiving the Connect to Node Request message, the first node should try to connect to the second node. But before doing it, the node should verify if the second node is not itself or if it is not already connected to it. If one of these situations is verified, the first node should reject the request.

If the first node is a probe and it is not connected to the second node, it should send a Bye message (appendix A.2.37) to its super-probe informing that it will close the connection. Then it should close the connection and try to connect to the second node. However, if the probe is not able to connect to the second node, it should reconnect to the previous super-probe. If the first node is a super-probe the request will not be accepted if its maximum number of connections has been reached.

After trying to connect to the second node, the first node should send a Connect to Node Response message (appendix A.2.36) to the client that issued the request (Figure 36). This message is used to inform the client if the first node accepted to process the request and if

it was or not successfully performed. Opposite to all response messages, the Connect to Node Response message is not routed through the reverse path, but must be flooded towards the client using a different Message ID than the one used by the corresponding request (a process similar to the one described in section 4.2.1). This must be done because the first node may be a probe that has connected to a super-probe which didn't receive the Connect to Node Request message sent by the requesting client. Note however that the Connect to Node Response message must include the Message ID of the Connect to Node Request message so that the client can correlate the response with the request it sent.

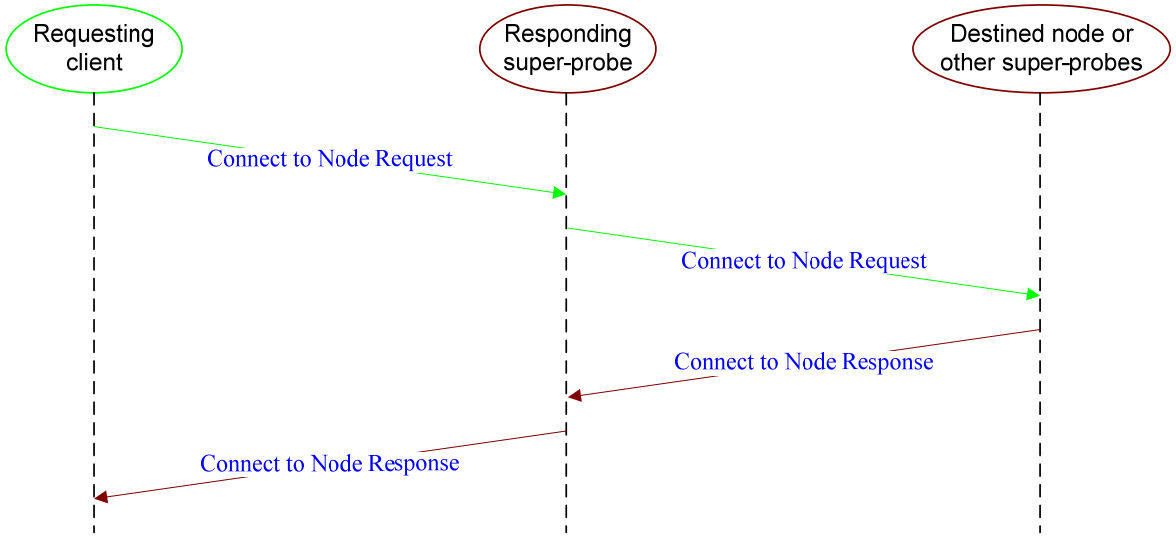

Figure 36. Messages exchanged during a connect to node request process.

4.9.4 Node's File List

Implementations of the DTMS-P2P protocol must allow a user to obtain the information about all the files a remote node is sharing to the network. This information should be provided in a file containing the name and the size in bytes of the files the node is sharing. This file will be called Node's File List (NFL). At any time, a user may use a client element to obtain the NFL directly from the required node.

4.9.4.1 Node's File List format

We have adopted XML as the NFL format because of its readability which facilitates the exchange of this file between the elements of different implementations of the DTMS-P2P protocol. This file must be defined using the following Document Type Definition (DTD):

```
<!DOCTYPE FileListing [
  <!ELEMENT FileListing (Directory*)>
  <!ELEMENT Directory (Directory*, File*)>
  <!ATTLIST FileListing nodeAddress CDATA #REQUIRED>
  <!ATTLIST FileListing groupID CDATA #REQUIRED>
  <!ATTLIST FileListing mode CDATA #REQUIRED>
  <!ATTLIST Directory name CDATA #REQUIRED>
  <!ATTLIST File name CDATA #REQUIRED>
  <!ATTLIST File size CDATA #REQUIRED>
]>
```

Figure 37. Node's File List XML DTD.

In this XML DTD the “FileListing” element comprise the information about the files a node is sharing to the network. The “Directory” element represents a directory with other directories and/or files being shared by the generating node. The “File” element represents a shared file. The “FileListing” element comprises three attributes used to identify the generating node. They are the “nodeAddress” (the IP address of the generating node), the “groupID” (the node’s measurement group ID) and “mode” (the node’s operating mode – probe or super-probe). The “Directory” element has an attribute which represents the name of the shared directory (“name”). The “File” element has two attributes, one represents the name of the shared file (“name”) and the other represents its size in bytes (“size”).

Figure 38 illustrates an example of a NFL.

```
<FileListing nodeAddress="193.136.92.234:22360" groupID="00000000000000000000000000000000"
mode="super-probe">

  <Directory name="results">

    <Directory name="error">
      <File
name="00000000000000000000000000000000_193.136.92.234.21164_00000000000000000000000000000000_
193.136.92.234.22361_1182274244918_ping 193.136.92.234 -d.err" size="787" />
    </Directory>

    <File
name="00000000000000000000000000000000_193.136.92.234.21164_00000000000000000000000000000000_
193.136.92.234.22360_1182251089911_ping 193.136.92.234 -n 10.res" size="787" />
    <File
name="00000000000000000000000000000000_193.136.92.234.21164_00000000000000000000000000000000_
193.136.92.234.22360_1182260883321_ping 193.136.92.234 -n 10.res" size="787" />
    <File
name="00000000000000000000000000000000_193.136.92.234.21164_00000000000000000000000000000000_
193.136.92.234.22360_1182273636063_ping 193.136.92.234 -n 10.res" size="787" />

  </Directory>

</FileListing>
```

Figure 38. Node’s File List example.

4.9.4.2 Node’s File List retrieval

A user must be able to configure a client to obtain a NFL from any node of the network. For this purpose, a client may be configured to obtain the list of nodes of a given measurement group using the mechanism described in section 4.4.2.2.2. A user can also enter manually the address of the node whose NFL he wants to download.

A client must connect directly to the selected node to request the download of the NFL. This file must be downloaded using HTTP using the same process as the one described for the retrieval of a super-probe’s Compiled Light Data File (section 4.8.3). But in this case the client should request for the download of the *FileList.xml* file instead of requesting for the *LightData.xml* file. The *FileList.xml* file name must be used to allow the remote node to identify that the requesting client wants to download its NFL.

A NFL must only be build after a user request its download and must be deleted from the node’s local storage after the file is completely uploaded. Using this procedure, a user will always receive the most recent information about the files the node is sharing to the

network. Due to the fact that a node may receive different NFL download requests at the same time, implementations must control the creation and deletion of the NFL.

After receiving a HTTP GET request message requesting for the node's *FileList.xml* file, the receiving node must build its XML NFL with the information about all the files it is currently sharing and send an HTTP response message to the requesting client. Using the Content-Length header of the received HTTP response message the client is able to determine the length of the NFL. The HTTP response message's Content-Type header must be equal to "fileList/xml".

After the completion of the NFL download, the client must store it in its download directory with a different name. The name of the NFL may be "*IP Address.Port_FileList.xml*", where IP Address and Port represent the IP address and port number of the generating node. Remember that a client may be used to download the NFL of different nodes. Whenever a client request again the NFL of a node which NFL was already downloaded before, the previous NFL must be replaced with the new one.

4.10 Performance security considerations

Besides the security constraints described in section 4.1, implementations of the DTMS-P2P protocol must implement other mechanism to guarantee the best performance of the system. In this section are described some of these mechanisms.

4.10.1 Preventing third-party denial of service

A lot of DTMS-P2P control messages directed from an unsuspecting party or a lot of connections requests could be used for denial of service (DoS) attacks. Thus, unless configured otherwise, super-probes should limit the number of connections they can accept from other elements. In addition, elements should only process messages received from another element, if they didn't receive a lot of messages from the remote element in a short period of time.

4.10.2 Resource use limitations

A DTMS-P2P node can consume resources of various kinds. The three most important kinds of resources are storage space capacity for storing test results, memory and network capacity. Any implementation of DTMS-P2P node must include technical mechanisms to limit the use of storage and memory capacity and to limit the use of network capacity. The default configuration of an implementation must enable these mechanisms and set the resource use limits to conservatively low values.

4.11 Summary

In this chapter we described the DTMS-P2P protocol and how the messages are exchanged between the elements of the system.

As described in this chapter, the DTMS-P2P protocol supports three security modes: unauthenticated, authenticated and encrypted. To guarantee the secrecy of exchange and in the authenticated and encrypted security modes, all the messages should be encrypted

using AES-CBC and authenticated using HMAC-SHA1. The security features should be used to prohibit theft of service and to provide secrecy of exchange.

The messages exchanged between the elements of the system must be forwarded, by the super-probes of the system, to other nodes connected to them. Each receiving node should only process the message if it is destined to it, to its measurement group or to all nodes of the system. The destination nodes should answer to the received requests and these responses must be routed back to the requesting element, using the same path the requests were received.

To connect to the network the elements of the system must try to connect to another node already connected to the network. This information is initially provided by the element's administrator but the elements should also exchange the information about known nodes in the process of connecting to each other. This information is used during connection set-up process.

When a user wants to configure a test session at a remote note, he must use a client to obtain the required information from the node where the test session is to be performed. After the end of a test session the user may retrieve the results of the test session by ending a query to the network.

After generating a new Heavy Data File, with the results of a performed measurement test, the generating node may replicate the file at other nodes of the network. To do so, the node must try to find possible locations where the file can be replicated and then request these nodes to download the file from its storage space or from other nodes where the file was previously replicated.

In this chapter we also described the overall download process used in the DTMS-P2P protocol. The file download process is performed using HTTP directly between the storing node and the receiving node, outside the DTMS-P2P network. Thus, the two involved elements must open a TCP connection between them and then process the file transference, from the storing node to the receiving element. During the file transference process the involved nodes must have attention to the fact that one of them may be firewalled. Only not firewalled nodes can receive connections requests from other nodes. Thus, case the node storing the file is behind a firewall and the receiving node isn't, this last one should send the remote node a request to "push" the file to it. When the file is stored in multiple nodes of the system, the DTMS-P2P should allow an element to download it from multiple sources. This operation may improve the download process.

The super-probes of the system should periodically generate a Light Data File (LDF), with the RTT statistics to each node connected to them, and forward the file to all super-probes of the overlay network. This process should be accomplished using the DTMS-P2P network to guarantee that this information can reach the majority of the super-probes of the network. Whenever required, a client can obtain, from any super-probe of the network, the compiled Light Data File (CLDF) with the information stored in the most recent super-probe's LDF plus the information stored in the latest LDFs of other super-probes.

As described in this chapter, a user may use a client to end different types of requests to remote nodes of the overlay network. Among these requests the user can configure the

client: to obtain the available resources at a remote node; to request a node to connect to another node of the network; to request a node to replicate and/or delete a given file; or to request a node for the information about all files the node is sharing to the network.

In this chapter we also described some security issues related to the performance of the DTMS-P2P protocol. The DTMS-P2P implementations should prevent the denial of services (DoS) attacks and must include mechanisms to limit the use of resources capacity (mainly storage space, memory and network capacity).

Chapter 5. DTMS-P2P versus File Sharing Applications

In this chapter we will compare the similarities and differences of the file search and download mechanisms proposed in this dissertation with the ones used by some file sharing applications. We will also compare DTMS-P2P with the P2P systems based in the Distributed Hash Table (DHT) method. In this last case we will pay special attention in the comparison between the overlay network construction, message routing, file storage, search and replication mechanisms implemented by these last systems and the ones we propose in this work.

Because there are a lot of file sharing applications and each one implementing different approaches we decide to only compare the DTMS-P2P system with an application that implements a similar approach in comparison to the one we use, the LimeWire file sharing application [LimeWire], and with the applications implementing the BitTorrent protocol which uses a totally different approach. We compare DTMS-P2P with the P2P systems implementing the DHT method in general.

In section 5.1 we will compare the DTMS-P2P and the LimeWire systems and in section 5.2 we will compare the DTMS-P2P and the BitTorrent systems. Then in section 5.3 we will compare the DTMS-P2P system with the P2P systems based in the DHT method. Finally, in section 5.4 we will give a summary of the chapter and present the main conclusions.

5.1 LimeWire

The LimeWire [LimeWire] file sharing application runs over the Gnutella network and, as DTMS-P2P (section 4.6), implements the same file search mechanism of the Gnutella protocol. Moreover, it implements multi-source downloads too, but with some differences in relation to the DTMS-P2P system.

To process multi-source downloads the LimeWire P2P application splits the content of the file to be downloaded in a set of regions of three different main categories [LimeWire2002]:

- *Black regions:* regions that have already been downloaded to disk;
- *Grey regions:* regions that are currently being downloaded;
- *White regions:* regions that have not been assigned to be downloaded yet.

As DTMS-P2P (section 4.7), the LimeWire application uses a split/steal algorithm as will be described next. For each available source is assigned a process (called downloader) responsible to download from the respective source. Each downloader executes the following actions:

i. Establishes a TCP connection to the respective source, using a PUSH message whenever required (as in DTMS-P2P – section 4.7.2). In case of failure the

downloader aborts and terminates; the correspondent source is not used anymore;

ii. Chooses a region of the file to download. If there is a white region, the downloader tries to download it. Otherwise it tries to steal, from another downloader, part of a not downloaded segment of a grey region. In case, there aren't other regions to be downloaded the downloader aborts. If this last situation is verified, the correspondent source may be used later whenever required;

iii. In case the downloader is able to get a region to be downloaded it sends an HTTP request for the given region and downloads the file. If the download terminates prematurely the downloader updates the list of regions to be downloaded accordingly. If the request fails, it aborts and terminates. In this last case, the source is not used anymore.

LimeWire uses two different approaches to choose the region of the file to download (grey or white region), depending on the version of the HTTP protocol being used. For HTTP/1.0 each request requires a new HTTP connection. Thus, in this case, to minimize the number of requests needed, a downloader assigns to itself an entire white region if available or half a grey region otherwise. Only one downloader at a time can execute the action (ii) above. When a downloader steals half a grey region from another one, the first downloader will close the connection when it gets halfway through the respective grey region. On the contrary, for HTTP/1.1 the LimeWire application defines a small fragment of N bytes to be downloaded at a time. In this case, a downloader chooses the first N bytes of a white region if available or the last N bytes of a grey region otherwise. Thus, if a file is being downloaded from two sources A and B, at the same speed, the file will be downloaded in the following order $ABABAB\ldots$ This approach has two major advantages: (i) it improves the file preview functionality, as the file is typically downloaded from beginning to end; and (ii) it reduces the delay provoked by slower sources.

In both cases described above, the split/steal process is performed whenever a downloader completes before the other ones.

At the split/steal process LimeWire, as DTMS-P2P, has a provision where a faster downloader can steal an entire region from a stalled downloader.

As DTMS-P2P, at the LimeWire P2P application different downloaders write the content of the file to be downloaded to the same temporary file which avoids having to reassemble fragments at the end of the download.

When comparing the download mechanism proposed in this dissertation (section 4.7) with the one implemented by the LimeWire P2P application we can notice that, although very similar, they differ in some aspects as will be described next.

DTMS-P2P initially computes the length of the fragments of the file to be downloaded from the available sources based in the number of available sources. The file is initially divided in a number of fragments equal to the number of available sources and with equal size. Each fragment is downloaded from one of the available sources. Thus, the overhead

required to download these fragments will be, in most cases, less than the overhead obtained in LimeWire. Moreover, DTMS-P2P defines the minimum size of a fragment to be downloaded from each source (minDownloadSize – default 1 MB). This allows the selection of faster sources when the size of the fragments, computed based in the number of available sources, is less or equal to minDownloadSize. Due to this, the file download will be faster.

As LimeWire, at DTMS-P2P whenever there is a free source, it should be used. However, unlike LimeWire, to be able to use the free source the downloader element should identify the slower source currently being used. If the element has to download yet more than the minDownloadSize from this source, the remaining bytes to be downloaded may be divided between this source and the free one. When splitting the remaining bytes to be downloaded, DTMS-P2P has in consideration the time required to download the bytes from both sources. To improve the download process, it divides the remaining bytes to both sources in a way that they will spend the same time to download them. This behavior prevents the element to perform the split/steal process many times, which improves the overall download process. This behavior is not verified in LimeWire.

5.2 BitTorrent

Let's now compare the applications implementing the BitTorrent peer-to-peer file sharing protocol [BitTorrent2007] [Cohen2003] with the proposed system. There are a lot of applications, called BitTorrent clients, implementing the BitTorrent protocol. Using a BitTorrent client, users are capable of preparing, downloading and sharing files to the network.

In the BitTorrent protocol any computer running an instance of a BitTorrent client is called a peer. For a given file, peers can generally be classified into downloaders and seeds. Seeds are peers who have the complete copy of the file (may be the original peer or other peers that successfully downloaded the file). Downloaders are peers who are downloading fragments of the file from other peers and may be sharing some fragments too (downloaded from other downloaders or seeds). After downloading a complete copy of the file, downloaders are upgraded to seed status.

Another element of the BitTorrent protocol is the tracker. This entity is responsible to coordinate the file distribution by maintaining a list of the clients currently participating in the file sharing. These clients are peers who are currently sharing the file or one of its fragments. The tracker only manages connections, it does not have any knowledge of the contents of the files being distributed, and therefore a large number of users can be supported with relatively limited tracker bandwidth.

To share a file, a peer must first split the file into fragments and compute a hash on each one. These hashes are used by clients to verify the integrity of the data they receive. After computing the hashes, a peer creates a file called "torrent" file. This file is a small file which contains metadata about the file to be shared (file name, file length, fragments length and fragment hashes), and about the tracker responsible to manage the distribution of the shared file.

The original peer must place the “torrent” file in a web server and publish a link to it in a website or elsewhere. Peers that want to download the file must first obtain the “torrent” file for it using the published link, and then open it using a BitTorrent client. With the information stored in this file, the client must connect to the required tracker to obtain the information about the peers from where the fragments of the file can be downloaded (the original seed or other peers that have already retrieved the file or some of its fragments). After downloading a given fragment and after they have checked the correspondent hash stored in the “torrent” file, peers report it to the tracker to start sharing it to the network too.

As described above, in the BitTorrent protocol, a file is not shared only by the original peer, receiving peers also supply data to newer recipients. Therefore there is a distribution of the costs of hardware, storage and bandwidth resources among all storing peers. This significantly reduces the load of the original peer and dependence upon any given individual source as well as provides redundancy against system problems.

During a file download, when selecting which fragment to start downloading next, peers generally first chooses fragments shared by fewest peers. This technique is called “rarest first”. It ensures the replication of the rarest fragments as quickly as possible thus reducing the risk of them getting completely lost if the peers originally sharing them stop sharing. A random fragment selection approach may also be used. This last approach increases the opportunity to exchange data, which is only possible if two peers have different pieces of the file.

The BitTorrent protocol uses HTTP for file downloads, however the BitTorrent download differs from the standard HTTP download, used by many web browsers, in several aspects:

i. BitTorrent makes many small P2P requests over different TCP sockets, while in a standard HTTP download typically is made a single HTTP GET request over a single TCP socket.

ii. BitTorrent downloads in a random or “rarest first” approach that ensures high availability, while in the standard HTTP files are downloaded in a contiguous way.

The BitTorrent protocol presents some inherent limitations:

i. the tracker is a single point of failure in the network. If a tracker fails, it is no longer possible for new peers to join the network or for existing peers to discover each other;

ii. the tracker is a bottleneck because the scalability of a BitTorrent network largely depends on the network capacity of the tracker.

When comparing the BitTorrent protocol with the DTMS-P2P protocol the main first difference that we observe is that BitTorrent protocol does not implement the file search mechanism. To share a file or group of files, a peer uses the “torrent” file. Peers that want to download the file must first obtain the “torrent” file for it, to get the information needed to download the file. When comparing the download mechanism implemented by the

BitTorrent protocol with the one implemented by DTMS-P2P (section 4.7), we can verify that both mechanisms are completely different. Moreover, BitTorrent deals well with files that are in high demand and newer files (files only shared from a few number of peers). The applications implementing the BitTorrent protocol have this advantage because the file is cut into fragments of fixed size and each fragment can be shared by the peers that previously downloaded them. On the contrary, the nodes of the DTMS-P2P protocol only share a file after it is completely downloaded.

Alternatively to the method described above, in a BitTorrent trackerless system (decentralized tracking) every peer acts as a tracker. This alternative scheme is implemented by the BitTorrent [BitTorrent], µTorrent [µTorrent], BitComet [BitComet] and KTorrent [KTorrent] clients through the distributed hash table (DHT) method [Balakrishnan2003] [Tanenbaum2007]. Azureus [Azureus] also supports a trackerless method but it is incompatible with the DHT offered by all other supporting clients.

5.3 Peer-to-peer systems based in Distributed Hash Tables

P2P systems based in the Distributed Hash Table (DHT) method [Tanenbaum2007] comprise many differences when compared with the traditional first-generation peer-to-peer architectures. In the first P2P systems the file search mechanism was based in a database maintained in a central server and which mapped file names to sources storing the files. This approach had inherent scalability problems and the central server was a single point of failure. In succeeding P2P systems (for example Gnutella protocol version 0.4 [Gnutella04]) the central server was removed to avoid the single point of failure problem. In these systems, all nodes share files in the network and act as a server. To search for a file, a query message is flooded to all nodes of the network and only the nodes sharing the required file answer to the request. Still, this approach doesn't scale well due to the large bandwidth consumed by broadcast messages and due to the load consumed at the many nodes that must handle these messages. To minimize this problem, a hierarchical approach was introduced in which some nodes are selected to act as an ultrapeer which are nodes that manage the exchange of messages between the nodes connected to them and other nodes of the network (for example, Gnutella version 0.6 [Gnutella06], which was used as the basis for our implementation). As will be described next, the DHT introduced a different approach to find files in a P2P system in a scalable manner without any centralized servers or hierarchy.

5.3.1 Overlay network and message routing

The DHT overlay network is build by assigning a unique node ID to each participating node and by the connections between nodes with consecutives IDs. The node IDs are *k*-bit identifiers belonging to an abstract large identifier key space, such as the set of 128-bit or 160-bit identifiers. The nodes are arranged by their IDs in a particular manner as, for example, in a ring where the IDs are in increasing order clockwise around the ring (Figure 39). The connections between nodes of the system are usually UDP based connections. To join the system a node must first contact with a node already connected to the overlay network to negotiate its node ID and to find its place in the network. Node IDs are chosen randomly and uniformly so peers who are adjacent in node ID can be geographically away from each other. After the selection of the node ID the new node can connect to the nodes which IDs are closer to its own to inform them that it is a new node. Nodes of the network

must keep a table with the information about their closest neighbors in terms of the key space (route table). This information must be updated whenever a new node connects to the network.

The route table is used to forward messages to destinations. Messages are usually destined to a node with a given node ID and must be sent, by the original node and all receiving ones, to the node in the route table with the closer ID (smallest ID greater or equal to the ID sent in the message). The message is forwarded from node to node through the overlay network until it reaches the single living node with smallest ID greater or equal to the ID carried in the message (Figure 39). This last node must process the request received in the message. This style of routing is sometimes called key based routing.

To maintain the overlay network and keep their route tables updated, the nodes periodically send messages to their neighbors to verify if they are still active. If a node does not respond the requesting node updates its route table and tries to connect to the node which ID is the succeeding ID in comparison to the ID of the node that left the system.

The key space partitioning, message routing schema and overlay network components described above are the principal ideas common to most DHTs. However many designs differ in the details. The first DHTs implementations where CAN [Ratnasamy2001], Chord [Stoica2001], Pastry [Rowstron2001] and Tapestry [Zhao2004]. In the Figure 39 we present an example of the DHT method implemented in Chord.

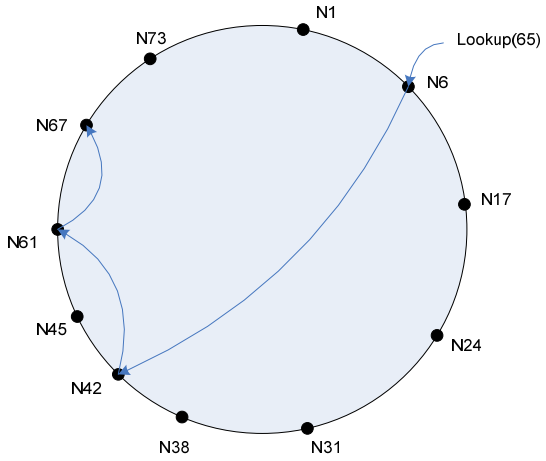

Figure 39. Chord DHT.

In the Chord and Pastry DHT implementations the nodes in the network are arranged in a circular identifier space and have similar characteristics as the ones described above. However, Pastry introduces some differences in the routing procedure.

To route messages a Pastry node first try to find a node ID that shares with its message ID a prefix that is at least one digit (or *b* digits) longer than the prefix shared with the ID of the current node. To do so, each node maintains three tables: routing table, leaf table and neighborhood table. To attempt to minimize the distance traveled by messages, each entry in the routing table points to one of potentially many nodes close to the present node according to a scalar proximity metric (for example, the number of IP routing hops or the round-trip time delay) instead of the key space proximity (numeric ID difference). The leaf

table consists of the *n* closest peers by node ID in each direction around the circle and is the first table used to find a match when forwarding a message. The routing table is only used if a match is not found in the leaf table. The neighborhood table is a set of *x* nodes that are near the present node, according to the proximity metric. It is not usually used in routing, but is useful for routing table updates when the state of the system changes (node addition or recovery). If a match in terms of the configured number of prefix is not found in both the leaf and routing tables, the message is forwarded to a node which ID, stored in one of the three tables, shares a prefix with the message ID at least as long as the local node, and is numerically closer to the message ID than the present node's ID.

In the Content Addressable Network (CAN) the peer nodes are placed in a virtual, d-dimensional Cartesian coordinate space.

The main differences that we can identify when comparing the DTMS-P2P system with the P2P systems based in the DHT method, in terms of the overlay network and message routing, are:

- The DTMS-P2P system is based in a hierarchical architecture (section 4.3) totally unlike the architecture assembled by the DHT based P2P systems. Due to its hierarchical structure more complex mechanisms are required to maintain the DTMS-P2P overlay network;

- When a node tries to connect to a DTMS-P2P network it first tries to connect to nodes that are geographical near to its location. This is not a main concern in most DHT systems which may lead to situations where messages unnecessarily travel long distances to reach their destination. To prevent this problem Pastry resorts to routing tables that account for the distance among nodes;

- The DTMS-P2P overlay network is based in TCP connections, thus the nodes are able to immediately identify when a node connected to them leaves the network. As a consequence, the DTMS-P2P nodes do not need to periodically send messages to their neighbors to verify if they are still alive as is usually required in DHT based systems;

- The nodes of the P2P systems based in the DHT method usually maintain a routing table with the addresses of some nodes to which messages can be sent to reach their destination. In the DTMS-P2P system nodes do not keep a routing table (unless to root response messages back to the requesting element – section 4.2) since messages are usually flooded to the network. The flooding process may be disadvantageous because it may occupy large bandwidth since messages are flooded to a lot of unnecessary nodes and these nodes are required to process messages not destined to them. However, messages are more likely to reach their destination even if the topology of the network changes during the flooding process. This is not guaranteed in the DHT method because a node, supposed to forward a given message, may leave the network at any time without previous alert and without being noticed. In case of failure, the originating node must resend the message. The routing mechanism implemented in the DHT method is faster than the flooding mechanism.

5.3.2 File storage and search mechanism

In DHT networks, the filenames are hashed to a *k*-bit key (where *k* is the key space size in bits) and stored at the node with ID closer and greater or equal to this key value. Alternatively, instead of storing the file at the node, a pointer to the node where the file is originally located can be placed. To search a file, again the filename is hashed to a *k*-bit key and routed towards the node with ID closer and greater or equal to the key value, where the file is stored. The messages to put a file on a node and to retrieve a file from a node are routed as described in previous section.

Whenever a new node connects to the DTH network, it must be checked if this node should be the one to store one or more files presently stored in its neighbor with closest higher ID. If so the file or files should be transferred to the new node. This is a bandwidth-intensive task that can occur frequently if the network has high rates of nodes arrivals and failures. The DTMS-P2P system has not this problem since the storage and retrieval is not based in a key space partitioning scheme as it is in the DHT method.

DHTs only directly support exact-match search, rather than keyword search, although this functionality can be layered on top of the DHT. The DTMS-P2P data search mechanism provides both exact-match search and keyword search.

Due to the key based search mechanism implemented by the DHT method queries are likely to be routed through the network without needing to visit many peers resulting in faster queries. The DTMS-P2P system uses a flooding search mechanism which may require that queries have to travel a lot of nodes to reach their destination. In addition, this mechanism may lead to poor network performance since a lot of nodes receive and have to process messages not destined to them.

5.3.3 File replication

The DHT method also allows P2P systems to replicate the files at different nodes connected to the overlay network. DHT based P2P systems supporting replication have been proposed in [Druschel2001] and [Plavec2004]. Here, if *n* replicas are required, they will be stored in the *n* consecutive nodes with IDs greater or equal to the ID of the file to be replicated. Thus, the replication mechanism implemented by the DHT method does not have in consideration the load and resources available at the nodes where the replicas are to be created. These nodes are selected having in consideration the key ID of the file to be replicated and the node ID. On the contrary, in the DTMS-P2P system the locations where a file should be replicated are selected having in consideration their available storage resources and network speed.

The DHT P2P distributed system proposed in [Plavec2004] tries to guarantee that the number of replicas in the network is kept constant. This is not possible in the DTMS-P2P system since nodes are not aware of the nodes where the replicas are created and are not able to identify when a node leaves the network if was not directly connected to the disconnecting node.

5.4 Summary

There are different approaches used by different file sharing applications to share files among peers. In this chapter were compared the similarities and differences of the file search and download mechanisms proposed in this dissertation, with the ones used by the LimeWire file sharing application and with the ones used by the applications implementing the BitTorrent protocol. It is also made a comparison between the DTMS-P2P system with the P2P systems implementing the Distributed Hash Table (DHT) method.

The file search mechanism implemented in the LimeWire application is the same implemented in DTMS-P2P. This mechanism is described by the Gnutella protocol.

The download process used by the LimeWire application is very similar to the one proposed in this dissertation. The main difference resides in the fact that LimeWire does not have in consideration the number of sources from where it can download a file, to compute the size of each block of information (fragments) to be downloaded from the available sources. For HTTP/1.0, a downloader tries to download the entire file. Subsequent downloaders use the split/steal algorithm to download other fragments of the file. For HTTP/1.1, small blocks (of fixed size) of the file content are downloaded from different sources, from the beginning to the end of the file. The LimeWire has this behavior to improve file previews, since the file is almost continuously downloaded from the beginning to the end.

When splitting a fragment being downloaded by a given downloader, to be divided between this downloader and a free one, to improve the download process, DTMS-P2P divides the remaining bytes to both sources in a way that they will spend the same time to download their piece of the fragment. This behavior prevents the elements to perform the split/steal process a lot of times. This behavior is not verified in the LimeWire. LimeWire divides the section in two equal pieces instead.

The download mechanism implemented by the DTMS-P2P system is completely different from the one implemented by the BitTorrent protocol. Moreover, the traditional BitTorrent protocol does not implement the file search mechanism. In this protocol, to share a file or group of files, a peer first creates a “torrent” file and places a link to this file on a website or elsewhere. Peers that want to download the file first obtain a “torrent” file for it, and connect to the specified tracker (the computer that coordinates the file distribution) which tells them from which other peers to download the fragments of the file. After downloading a given fragment a peer may start sharing it to other peers. On the contrary of the BitTorrent protocol, the nodes of the DTMS-P2P protocol only share a file when it is completely downloaded.

The P2P systems based in the DHT method implement a completely different approach to build the overlay network, share and retrieve files from the network and the implemented replication method has many differences in comparison to the one implemented by DTMS-P2P. The file search mechanism of the P2P systems based in the DHT method provides a lookup service very similar to a hash table. It is based on the storage of (key, data) pairs and you can look up the data if you have the correspondent key. The implemented replication mechanism is based in the IDs attributed to the nodes and the file keys in the key space and does not have in consideration the available resources at the storing nodes.

Chapter 6. DTMS-P2P Implementation

In this chapter we present a versatile and easily manageable implementation of the DTMS-P2P system described in Chapter 3 and Chapter 4, which was also designated by DTMS-P2P. The DTMS-P2P system was implemented in Java language using Eclipse IDE [Eclipse]. This language was used because it presents a set of favorable characteristics, like semantic simplicity, portability and a set of classes that greatly simplify the construction of distributed applications.

In section 6.1 is presented the package structure of the implemented system and in section 6.2 we will list and describe the implemented classes. Finally, in section 6.3 we will give a summary of the chapter and present the main conclusions.

6.1 DTMS-P2P package structure

To implement the system the following 2 packages were built: *dtms_p2p* and *util* packages; and the following 4 sub-packages of package *util* were built: *aes*, *hmac*, *pbkdf2* and *xml* packages. Below there is a representation of this structure:

- *dtms_p2p*
- *util*
  - *aes*
  - *hmac*
  - *pdkdf2*
  - *xml*

In the *dtms_p2p* package are implemented all the classes directly related to the DTMS-P2P protocol implementation. In the *util* package are implemented some auxiliary classes used by the classes of the *dtms_p2p* package. The description of the classes implemented in both packages will be presented in the next section. In the *aes* package is implemented the class *AES_CBC* used to perform data encryption using AES (Advanced Encryption Standard) in Cipher Block Chaining (CBC) mode [AES]. In the *hmac* package is implemented the class *HMAC_SHA1* used to implement the HMAC-SHA1 [RFC2104], a mechanism for message authentication using cryptographic hash functions. In the *pbkdf2* package is implemented the class *PBKDF2* used to implement the Password-Based Key Derivation Function PBKDF2 (PKCS #5) as described in [RFC2898]. In the *xml* package is implemented the class *XMLParser* used to access XML documents using Document Object Model (DOM) API (appendix A.4).

6.2 DTMS-P2P implemented classes

In this section we will present a basic description of the Java classes developed to implement the proposed protocol.

6.2.1 Classes implemented in the *dtms_p2p* package

The structure of the system implemented in the *dtms_p2p* package, is based on three levels: Messages level, Entities level and Protocol level. At the Messages level, a set of classes

corresponding to each message that is exchanged in the DTMS-P2P protocol and some auxiliary classes were developed. At the Entities level, a set of classes was developed in order to implement the two elements (node and client) of the DTMS-P2P architecture. At the Protocol level a set of classes where build to implemented the protocol and to implement auxiliary functions and objects.

Bellow we give a more detailed description of the classes implemented in the *dtms_p2p* package for the levels identified above.

6.2.1.1 Message level

At the message level, the main class ***Message*** is the basis of the implementation of all the messages exchanged between the elements of the system, excepting the Server Greeting, the Server Start and the Setup Response messages. Thus, this class contains all methods (basic functions) for manipulating and formatting the different messages involved in the protocol. For each one of these messages we developed a class that derives from the *Message* class and which name corresponds to the name of the respective message. The Server Greeting, the Server Start and the Setup Response messages were also implemented in classes that have the same name of the respective messages.

Sending a message always implies transferring all its bytes to a buffer and send it through a socket, so all the classes implementing the messages of the protocol include a constructor to build the message and a method to transfer all its bytes to a buffer (*messageToBuffer*). In the same way, the reception of a message implies its reception through a socket and transferring information from the reception buffer to the respective message format. All these classes include a constructor that implements this functionality.

6.2.1.2 Entities level

At the Entities level, the main class ***DTMS_P2P_Element*** is the basis for the derived classes *DTMS_P2P_Node* and *DTMS_P2P_Client* used to implement the node (probe and super-probe) and the client elements, respectively. It was defined in order to group in the same class all objects and methods that are common to the two classes that implement the different elements of the DTMS-P2P architecture.

The ***DTMS_P2P_Node*** class is a subclass of the *DTMS_P2P_Element* class and implements the command line version of the node element of the protocol. It is used to represent the probe and super-probe elements of the DTMS-P2P architecture. This class includes all the objects and methods that are necessary to manage communications with other elements of the network. It includes, among others, a method to control the number of connections the node can make or accept and a method to control the process to receive messages from the network. The node's configuration can be done from the command line using a set of possible options.

The ***DTMS_P2P_NodeGUI*** class implements the graphical user interface (GUI) of the DTMS-P2P node element. It is a simple graphical interface that users may use to configure and run a DTMS-P2P node. Using this interface, a user can change a set of options of the node's configuration at any time while the node is running. This is not possible using the command line implementation of the DTMS-P2P node (*DTMS_P2P_Node*). Moreover,

this GUI provides to the user the information about the current node's status; a set of information about all the connections the node currently has active (node's LAC – section 4.3.3); the information currently stored in the node's Cache Of Known Nodes (node's CKN – section 4.3.2); the list of files the node is sharing to the network (measurement test results); in case the node is a super-probe, the list of stored Light Data Files (section 4.8); and a console where is printed all log messages. All these information are updated at run time. In addition, the GUI allows a user to configure the node to connect to a remote node or to remove any existent connection, which allows a user to configure the topology of the network. The user may also open any file (Heavy Data Files or Light Data Files) directly using the GUI.

The ***DTMS_P2P_Client*** class is a subclass of the *DTMS_P2P_Element* class and implements the command line version of the client element of the protocol. With this element a user may configure test sessions and retrieve the results produced by these tests. This class includes all the objects and methods that are necessary to establish the communication to the network, to configure test sessions at remote nodes and to retrieve the produced results. It includes, among others, a method to obtain information about the existent measurement groups and nodes at the network, methods to obtain information about a given monitoring module a remote node supports, a method to obtain information about all the monitoring modules a remote node supports, a method to allow a local administrator to configure a remote node to process a given test session, a method to retrieve the results and methods to configure remote nodes to process a required action (file replication or deletion, available resources request, connection request, etc). The client configuration can be done from the command line using a set of possible options.

6.2.1.3 Protocol level

At the Protocol level, the main class ***DTMS_P2P_Protocol*** class is the most important class of the system. It implements the main functionalities of the DTMS-P2P protocol and is used to process all messages received by an element. It also includes all the required objects and methods used to handle the exchange of messages between all the elements of the system. Each message received should be processed in a separate process because the node should be able to process different messages at the same time. Thus, this class extends the Thread class.

The ***Authentication*** class is used to process the encryption/decryption of messages sent/received in an established connection, using the AES (Advanced Encryption Standard) in Cipher Block Chaining (CBC) mode. It is also used to process the message content authentication using HMAC-SHA1. Moreover, it is used to store the required information exchanged during connection set-up, between two elements wishing to use the authenticated or the encrypted security modes. This information will be used to process the encryption and authentication of messages exchanged between these elements (section 4.1).

The ***ByteArrayList*** class is used to store the history of the messages sent by a client (client's message list – section 4.2). It is a circular list. When the maximum number of records that can be saved in this list is reached, the older record is replaced with the new one. This class extends the *ObjectTable* class.

The ***CacheOfKnownNodes*** class implements the Cache of Known Nodes (CKN) of a DTMS-P2P element. This class includes all the objects and methods required to store and manage the CKN of a given element of the system. This list can comprise the addresses of the known probes and super-probes of the same measurement group of the respective element, or the addresses of the known super-probes and probes of other measurement groups of the system. This class extends the *DefaultHandler* class because it is used to parse the content of the element's File of Known Nodes XML file (section 4.3.2) using the SAX API (appendix A.4).

The ***Connection*** class implements the socket connection between two elements of the system. It contains a method to request a TCP socket connection to a remote IP address and methods that are needed to receive and send messages through the socket. The class also contains a set of methods and objects used to manage the connection to the remote element.

The ***ConnectionList*** class is used to implement the object used to store and manage the list of connections of a given element (section 4.3.3).

The ***ConnectionManager*** class was defined in order to group in a same class all objects and methods that are common to the two classes that implement an element outgoing connection process and incoming connection process, *OutgoingConnectionManager* and *IncomingConnectionManager* classes respectively.

The ***DownloadDescriptor*** class describes a particular file/NodeHit pair that together contains all the information needed to initiate and control a download of a given file from a remote node.

The ***Downloader*** class defines an auxiliary object used to download a specific section of a file from a particular host, and save the downloaded data to a specific file on disk (section 4.7.3). This class extends the Thread class and is used to process the file download in a new thread. An element downloading the content of a given file should not be blocked while processing the file download.

The ***FileHit*** class defines an auxiliary object used to store information of a given file, usually returned as a hit of a Query message (Query-Hit message, appendix A.2.28). It is used to store the information about the file name, file size and information about all nodes that are sharing the file. This class is used by an element to identify all the nodes sharing a given file at the network and from where the file can be downloaded.

The ***IncomingConnectionManager*** class is used to handle the incoming connection requests of a given DTMS-P2P element. It extends the *ConnectionManager* class and is used to control and process the connection requests a given element receives in a separate thread. A node and a client element of the system should always be listening for incoming connection requests. They may be incoming connection request received from remote elements trying to connect to them and or incoming connection requests to be used in file download processes (section 4.7). Clients must only accept incoming connection requests if they are to be used in a file download process when the storing node is firewalled or, for any other reason, cannot accept the client connection requests (section 4.7.2).

The ***LightDataGenerator*** class implements the node's light data generator. It is used to generate the light data of a given super-probe connected to the network (section 4.8). This process should be periodically performed in a separate thread. Thus, the class extends the Thread class. It includes all the required objects and methods to generate a super-probe's light data and forward it to all super-probes connected to the generating super-probe.

The ***ListOfNodeAddr*** class implements the list of node addresses of the Ping and Pong messages (appendix A.2.5 and A.2.6, respectively).

The ***MonitoringModuleDescription*** class implements an object, used to store the description of a given monitoring module supported by a given node (section 4.4.1). It stores the information about the supported monitoring module as, for example, the name of the monitoring module, the command to get the monitoring module's help or usage description, the list of option to save to a file and the list of restrictions the local administrator configured the node to require for the respective monitoring module.

The ***MultiSourceDownloader*** class is used to implement the multi-source download, when a file to be downloaded is stored in multiple sources at the network (section 4.7.3). This class includes all the objects and methods that are necessary to handle the overall download process, to download different sections of the file from different sources and to build the entire downloaded file. This class uses instantiations of the *Downloader* class to download the different sections of the file from the available sources.

The ***NodeHit*** class defines an auxiliary object used to store information of a given node that is sharing a given file to be downloaded. It is used to store the information about the node's IP address, upload speed and some other information.

The ***ObjectTable*** class was defined in order to group in a same class all objects and methods that are common to the classes that implement elements used to store a set of objects. The classes *ByteArrayList, ReplicationTable* and *RouteTable* extend this class.

The ***OutgoingConnectionManager*** class implements the outgoing connection process when an element is trying to connect to the network (section 4.3.4). This class includes all the objects and methods that are necessary to process the element's File of Known Nodes, in order to connect to a remote node of the network. This class extends the *ConnectionManager* class because the outgoing connection set-up process should be processed in parallel, in a different thread. An element should not be blocked during this process.

The ***RemoteNodeAddr*** class represents an object used to store the IP address of a remote node (*IP_Addr* class) and the average RRT obtained to the respective node. This class is used in the *CacheOfKnownNodes* class to sort an element's CKN using the RTT statistics obtained to the remote nodes.

The ***ReplicationRecord*** class implements the object used to store, in the node's Replication Table, all the information relative to a replication of a given file (section 4.5.2). Among the information it stores, it comprises the information about the number of replications requests made, the number of successful replications made and the information about other

nodes where the file has being successfully replicated. It is only used during the replication of a given file.

The ***ReplicationTable*** class defines the object used to store the information (*ReplicationRecord*) of all files the node is replicating (section 4.5.2). A node should be able to replicate more than one file at the same time. This class extends the *ObjectTable* class.

The ***ResourcesInformation*** class implements the object used to represent the information about the available resources at a given node. Some of the information stored in this object are the available free memory space, available free storage space and information about the transfer rates supported by the node. It is used to represent the objects of the list of resources information of the Resources message (appendix A.2.33).

The ***RouteTable*** class implements the object used to store the history of the messages a super-probe received and on which connections they were received (section 4.2). This is used to route back the responses to messages received and forwarded by a super-probe. Each record of the *RouteTable* is used to map the Message ID of a message to the connection where it was received. When the maximum number of records that can be saved in this table is reached, the older record must be replaced with the new one. This class extends the *ObjectTable* class.

The ***Section*** class implements the object used to represent a particular section of a file that should be downloaded from a given source (section 4.7.3). It includes all the required information to describe the required file's section.

6.2.2 Classes implemented in the *util* package

Bellow is given a more detailed description of the developed classes of the *util* package. The classes implemented in the *util* sub-packages were described in section 6.1.

The ***IP_Addr*** class implements the object used to represent the IP address and port number where an element is waiting for incoming connections in the DTMS-P2P network. It is used to represent both IPv4 and IPv6 addresses.

The ***Log*** class implements an object used to write the system log messages to a file and/or to the screen.

The ***Timestamp*** class implements the object used to obtain and process the information time of the system. It implements the timestamp object format as in [RFC1305]. A Timestamp object should comprises two values: an unsigned integer number representing the seconds elapsed since 0h on 1 January 1900 and an unsigned integer number representing the fractional part of a second that has elapsed since then. This class comprises the required objects and methods to compute the value of the timestamp based on these two unsigned integer values and the opposite situation.

The ***Util*** class defines the base methods used by all classes of the DTMS-P2P protocol. It implements some methods as the method to compute the free disk space, the method to get a node's local IP address, the method to convert a string to a byte array, the method to

convert an unsigned integer to long, a method to obtain the *ping* RTT statistics to a remote IP address, etc.

6.3 Summary

In this chapter was described the DTMS-P2P implementation used to test the proposed system. This implementation was developed in Java language using the Eclipse IDE.

Descriptions of the set of packages and of the set of classes build to implement the overall DTMS-P2P system were presented.

The implementation is based in two main packages: the *dtms_p2p* and the *util* packages. In the *dtms_p2p* package are implemented the main classes of the system. The structure of this package is based on three levels: Messages, Entities and Protocol. At the Messages level, a set of classes corresponding to the messages that are exchanged in the DTMS-P2P protocol and some auxiliary classes were developed. At the Entities level, a set of classes was developed in order to implement the two elements (node and client) of the DTMS-P2P architecture. At the Protocol level a set of classes where build to implemented the protocol and to implement auxiliary functions and objects. In the *util* packages are implemented some auxiliary classes and some sub-packages with other auxiliary classes used by the classes of the *dtms_p2p* package.

Chapter 7. Evaluation and Validation of the DTMS-P2P System

In this chapter we present the results of a set of experiments carried out in the command line version of our implementation of the DTMS-P2P system. The network used in the experiments is shown in Figure 40. The machine with IP 193.136.92.107 was running the client element and all the other machines were used to run a super-probe or a probe element, depending on the experiment being performed. All the elements used during the experiments were configured to support only the unauthenticated security mode. The RTT between the machines represented in Figure 40 was about 0.6 ms. The characteristics of the machines are presented in Table 3.

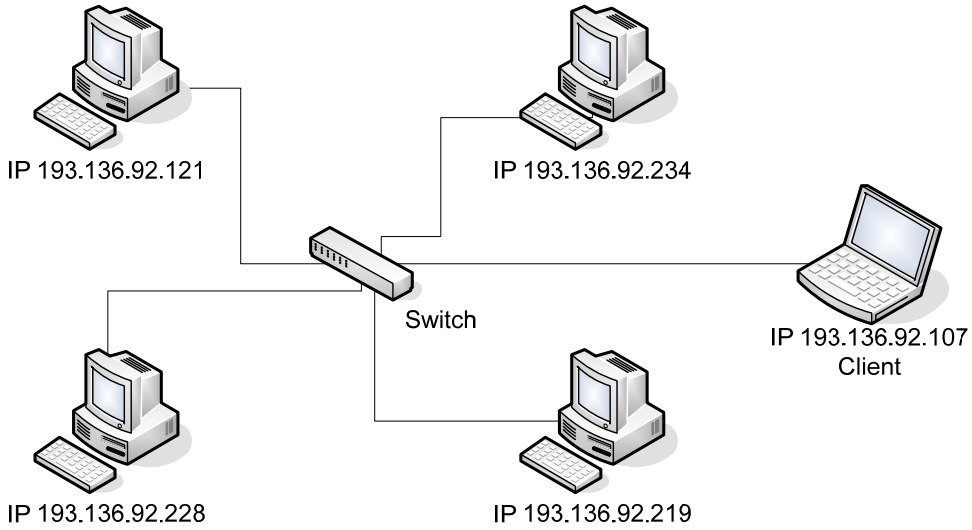

Figure 40. Network used to test the DTMS-P2P system.

Table 3 – Machines Characteristics

IP	OS	Network Card speed (Mbps)	Processor (GHz)	Memory (MB)
193.136.92.107	Windows XP	100	PM 1.8	1024
193.136.92.121	Linux	10	P4 2.4	512
193.136.92.234	Windows XP	100	P4 3.19	1024
193.136.92.228	Windows XP	100	P4 3.20	1024
193.136.92.219	Windows 2000	10	P3 0.300	256

Several experiments were performed. The first experiment measures the duration of a connection set-up process, when a node is trying to connect to the network (section 7.1). The second experiment measures the delay to receive the information about the Group ID of all existing measurement groups when a client is trying to discover which measurement groups exist in the network (section 7.2). The third experiment measures the delay to receive the list of nodes of a given measurement group (section 7.3). The fourth experiment measures the delay to receive the hits sent in response to a requested query (section 7.4). The fifth experiment measures the download speed of a file (section 7.5). This experiment was also performed to compare the behavior of a multi-source download and single source download. The sixth experiment (section 7.6) compares the download

speed obtained using the DTMS-P2P system with the download speed obtained directly from a web server using Internet Explorer (IE) and a download manager (FlashGet [FlashGet]) and the download speed obtained with the LimeWire application [LimeWire].

7.1 Connection set-up

To measure the duration of a connection set-up process (section 4.3.5) two experiments were performed:

1. The first experiment was performed to compare the set-up time obtained (i) when a probe connects to a super-probe of its measurement group with the one obtained (ii) when a super-probe connects to another super-probe of the same measurement group. Only two machines of the network of Figure 40 were used. In case (i) we executed a super-probe in one machine and a probe in the other one and in case (ii) we executed a super-probe in both machines. In both cases the responding super-probe was not connected to any other element and its Cache of Known Nodes (CKN – section 4.3.2) was empty.

2. The second experiment was performed to study the influence of the number of nodes in the connection set-up time experienced by a new node. In this experiment only three machines of the network of Figure 40 were used. Two machines simulated the overlay network, one running a super-probe and the other running 10 probes (in different ports) that were connected to the super-probe. The third machine was running the probe that attempts connection to the overlay network and for which the connection set-up time is to be measured. This probe only had the address of the remote super-probe in its File of Known Nodes (FKN – section 4.3.2) and didn't know the addresses of the other probes. All nodes were configured to belong to the same measurement group. The connection set-up time obtained in this case is compared with the one obtained in the first experiment, when a probe connects to a super-probe not connected to any other element.

In both experiments the connection set-up time is consider being the difference between the times at which the node trying to connect to the network starts and ends testing the addresses stored in its CKN. Thus, the connection set-up time does not include the delay observed while the node is reading the addresses provided in its FKN. Remember that an element must compute the RTT (round-trip time) to each address read from its FKN, to sort the provided list of node addresses in order of the obtained RTTs (section 4.3.2). Note that, in the case of a super-probe, the testing of the CKN includes the demotion negotiation process with each super-probe of its measurement group with which a connection is established (section 4.3.6.2).

For both experiments 10 test runs were performed and the obtained results and average values are presented in Table 4 and in Table 5, respectively.

Table 4 – Duration of a connection set-up process between two nodes of the same measurement group (p - probe; sp - super-probe)

Test Run	p-sp (delay in sec)	sp-sp (delay in sec)
1	0.125	0.187
2	0.125	0.157
3	0.125	0.156
4	0.141	0.203
5	0.125	0.157
6	0.187	0.172
7	0.140	0.172
8	0.156	0.157
9	0.157	0.156
10	0.125	0.172
Average	**0.1406**	**0.169**

Table 5 – Duration of a connection set-up process between a probe and a super-probe (connected to 10 probes)

Test Run	p-sp (delay in sec)
1	30.891
2	30.562
3	30.578
4	30.562
5	30.594
6	30.625
7	30.610
8	30.562
9	30.532
10	30.563
Average	**30.608**

From the analysis of the results in Table 4, it is possible to verify that the connection set-up between two super-probes of the same measurement group lasts longer than the connection set-up between a probe and a super-probe. This difference occurs because, in the case of a connection between two super-probes, due to the demotion negotiation process, there are more messages exchanged between the involved nodes. Remember that, as described in section 4.3.6.2, when two super-probes of the same measurement group connect to each other they must exchange information about their available resources. This information is then used to determine which super-probe(s) should maintain in super-probe mode.

The results in Table 5, in comparison to the results in Table 4, shows that the delay in the connection set-up process depends on the number of elements connected to the network. This can be explained by the fact that the connecting element will have to process a larger number of node addresses received from the node to which it is connecting to. As described in section 4.3.5, when an element connects to a remote element this last one sends to the requesting element a list with the addresses of nodes that exists in the network (information stored in the responding element's CKN). The bigger is this list the longer will take to process the node addresses in it. The requesting element must update its CKN with the received information (section 4.3.7). To do so, the element must compute the RTT to each new node address received from the remote node to be able to sort its CKN in order of the obtained RTTs. As can be verified in Table 5, the obtained delays are very similar to

the time needed to compute the RTT to 10 IP addresses. This delay is approximately equal to 30 seconds because, to compute RTTs, the node trying to connect to the network was configured to send 4 ICMP echo requests in intervals of one second. The first ICMP echo request is immediately sent, thus for each address the requesting node spends 3 seconds to compute the RTT. As there are 10 new node addresses to be added to the requesting node's CKN, it spends approximately 30 seconds to complete the CKN update. Therefore, the greater is the number of nodes connected to the network the bigger will be the delay for a new node to complete the connection set-up process. This delay is not an issue to the correct operation of the system because right after a node receives a Pong message from a remote node and this last one is accepting the connection request, the requesting node will be able to interact to the network while updating its CKN with the information received from the remote one.

7.2 Measurement group discovery

Two experiments were performed to measure the delay to receive the information about the Group ID of all existing measurement groups when a client is trying to discover which measurement groups exist in the network (section 4.4.2.1), after it connects to a super-probe. In the first experiment, besides the machine where the client was running, only two machines were used and they were running a super-probe. The super-probe in machine 193.136.92.121 was configured with Group ID ***a*** and the super-probe in machine 193.136.92.219 was configured with Group ID ***d***. These super-probes were interconnected to each other. In the second experiment, the machine where the client was running and all the other 4 machines were used. These last ones were running a super-probe. All super-probes were interconnected to each other and were from different measurement groups (Table 6). In both experiments, the client was configured to connect to the super-probe at machine 193.136.92.121 (Group ID ***a***). These experiments were configured to compare the delays obtained to receive the responses with the information about the Group ID of the measurement group of each super-probe and to verify if the delay to receive these responses does or not depend on the number of existing super-probes.

Table 6 – Super-probe's measurement group

Super-probe Host Machine IP	Measurement Group ID
193.136.92.121	***a***
193.136.92.234	***b***
193.136.92.228	***c***
193.136.92.219	***d***

For both experiments 10 test runs were performed and the obtained results and average values are presented in Table 7 and in Table 8, respectively.

Table 7 – Measurement Groups discovery (only 2 super-probes interconnected)

Test Run	Delay (sec)
1	0.091
2	0.084
3	0.095
4	0.099
5	0.093
6	0.093
7	0.101
8	0.096
9	0.090
10	0.099
Average	**0.094**

Table 8 – Measurement Groups discovery (4 super-probes interconnected)

Test Run	Response Receiving Order	Delay (sec)	Group ID
1	1st	0.089	***b***
	2nd	0.182	***c***
	3rd	0.193	***d***
2	1st	0.095	***b***
	2nd	0.180	***d***
	3rd	0.183	***c***
3	1st	0.108	***b***
	2nd	0.153	***d***
	3rd	0.155	***c***
4	1st	0.112	***b***
	2nd	0.158	***d***
	3rd	0.162	***c***
5	1st	0.109	***b***
	2nd	0.134	***d***
	3rd	0.138	***c***
6	1st	0.094	***b***
	2nd	0.132	***d***
	3rd	0.135	***c***
7	1st	0.100	***b***
	2nd	0.207	***d***
	3rd	0.209	***c***
8	1st	0.094	***b***
	2nd	0.132	***d***
	3rd	0.135	***c***
9	1st	0.104	***b***
	2nd	0.193	***d***
	3rd	0.196	***c***
10	1st	0.094	***b***
	2nd	0.132	***d***
	3rd	0.135	***c***
Average	1st	**0.0999**	
	2nd	**0.1603**	
	3rd	**0.1641**	

In Table 8 we give delays for the first, second and third responses that arrive at the requesting client. Note that both in Table 7 and Table 8 we do not include the delay on receiving information about the client's own measurement group. This is because upon reception of the client's request, the super-probe the client connects to, doesn't need to

inform the client what is its measurement group since the client knows the Group ID of its own measurement group.

When comparing the delay obtained to receive the response from the first super-probe in Table 8 with the one obtained in Table 7, we can verify that they are almost equal. Given this result, and taking into account the operation of the protocol, it is possible to anticipate that the delay to receive the information of the first *n* measurement groups of the system is independent of the number of super-probes in the system. This behavior was expected because the super-probe, to which the client is connecting to, must flood the client's request to all super-probes to which it is connected to in the same way, no matter the number of super-probes connected to it. This request is forwarded to all super-probes, one by one, until it is flooded to all super-probes. Each super-probe, upon reception of the client's request will send the information of its measurement group ID in response. Each receiving super-probe will also flood the received request message to the other super-probes connected to them.

From the Table 8 it is possible to verify that the responses are not always received in the same order (see, for example, Group ID column of test run number 1 in comparison to test run number 2). This is due to the flooding process and the differences in processing times of each super-probe in each test run, which does not guarantee that the first request arrives at a super-probe always in the same connection and that the requests arrives at the super-probes always in the same order.

7.3 Retrieval of the list of nodes of a given measurement group

A user can configure a client to obtain the list of nodes of a given measurement group that support a monitoring module he wants to use (section 4.4.2.2). This experiment was performed to measure the delay to receive this list of nodes and to compare the delays obtained from different measurement groups.

In this experiment, besides the machine where the client was running, only two more machines were used and they were running a super-probe. The super-probe in machine 193.136.92.121 was configured with Group ID ***a*** and the super-probe in machine 193.136.92.228 was configured with Group ID ***c***. These super-probes were interconnected to each other. The client was directly connected to the super-probe at machine 193.136.92.121 (Group ID ***a***).

For each measurement group 10 test runs were performed. In Table 9, we present the delay to receive the list of nodes of the measurement groups with Group ID equal to ***a*** and ***c***.

Table 9 – List of nodes of measurement group a and c

Test Run	Delay (sec) Measurement Group *a*	Delay (sec) Measurement Group *c*
1	0.006	0.022
2	0.012	0.019
3	0.009	0.017
4	0.007	0.018
5	0.007	0.015
6	0.007	0.017
7	0.006	0.017
8	0.012	0.016
9	0.006	0.016
10	0.006	0.019
Average	**0.008**	**0.018**

From the analysis of the results in Table 9 it is possible to verify that the delay to receive the list of nodes increase with the number of hops needed to reach the super-probe of a given measurement group (the delay obtained for the measurement group ***c*** is greater than the one obtained for the measurement group ***a***). This behavior is expected because upon reception of the request, the client's super-probe will first answer with its list of nodes supporting the required monitoring module. Then, it forwards the request to all super-probes connected to it and each receiving super-probe will have the same behavior. Thus, the more hops the request has to travel the higher will be the delay to receive the response.

7.4 Query-Hits reception

In this section we describe experiments performed to measure the delay to receive the hits of a query in a results search process (section 4.6). In this experiment, the machine where the client was running and all the other 4 machines were used. These last ones were running super-probes. All super-probes were interconnected to each other and were from different measurement groups (Table 6). The client was directly connected to the super-probe at machine 193.136.92.121 (Group ID ***a***). To perform this experiment a copy of the same file was placed in all super-probes of the network.

Two experiments were performed. In the first experiment we configured local searches of the file at the client's measurement group (Group ID ***a***). In the second experiment, we configured global searches for the file (all measurement groups (Table 6) of the network in Figure 40). These experiments were configured to compare the delays obtained to receive the hits of each super-probe and to verify if the delay to receive these hits depend on the number of existing super-probes.

For both experiments 10 test runs were performed and the obtained results and average values are presented in Table 10 and Table 11, respectively.

Table 10 – Query hits reception (local search at the client's measurement group, Group ID *a*)

Test Run	Delay (sec)
1	0.038
2	0.020
3	0.018
4	0.033
5	0.022
6	0.011
7	0.020
8	0.011
9	0.014
10	0.016
Average	**0.020**

Table 11 – Query hits reception (global search)

Test Run	Response Receiving Order	Delay (sec)	Group ID
1	1st	0.043	***a***
	2nd	0.154	***b***
	3rd	0.354	***d***
	4th	0.359	***c***
2	1st	0.030	***a***
	2nd	0.173	***b***
	3rd	0.181	***d***
	4th	0.373	***c***
3	1st	0.026	***a***
	2nd	0.211	***b***
	3rd	0.221	***d***
	4th	0.411	***c***
4	1st	0.017	***a***
	2nd	0.142	***b***
	3rd	0.145	***d***
	4th	0.342	***c***
5	1st	0.025	***a***
	2nd	0.199	***b***
	3rd	0.202	***d***
	4th	0.399	***c***
6	1st	0.016	***a***
	2nd	0.183	***b***
	3rd	0.186	***d***
	4th	0.382	***c***
7	1st	0.019	***a***
	2nd	0.214	***d***
	3rd	0.216	***b***
	4th	0.219	***c***
8	1st	0.020	***a***
	2nd	0.217	***d***
	3rd	0.220	***b***
	4th	0.231	***c***
9	1st	0.017	***a***
	2nd	0.180	***b***
	3rd	0.183	***d***
	4th	0.381	***c***
10	1st	0.018	***a***
	2nd	0.218	***d***
	3rd	0.222	***b***
	4th	0.233	***c***
Average	1st	**0.023**	
	2nd	**0.189**	
	3rd	**0.213**	
	4th	**0.333**	

From the analysis of the results in Table 10 and Table 11 it is possible to verify that in our experiments the delay to receive the first query hit, the query hit of the super-probe to which the client is directly connected to, is independent of the type of search used (local or global). This is only verified because we have only a few nodes connected to the overlay network. This may not be the case when there are a lot of nodes connected to the network since each super-probe first forwards a received query to all nodes connected to them and

only after verifies if they are sharing files that match the search criteria. This procedure is used to minimize the response delay of other elements in case a super-probe is sharing a lot of files. In the case of a global search, as in the experiment of section 7.2, and due to the same reasons, the responses are not received in the same order (see, for example, Group ID column of test run number 7 in comparison to the test run number 6). Note that the number of files a node is sharing may also influence the order of responses. In our test this is not the case since all super-probes were sharing the same amount of files (only 1 in this case).

7.5 Single source versus multiple source download speed

In this section we describe the experiments performed to measure the download speed (section 4.7) of the implemented system. The same configuration of section 7.4 was used. In Table 12 we present the measurement group of each machine (Figure 2) and the speed of its network card, classified as "fast" (100 Mbps) and "slow" (10 Mbps). To perform the experiments copies of 4 files with different sizes were stored in some or in all super-probes of the network of Figure 40. The client element was used to request their download. Two experiments were performed. The first experiment was performed to study the download speed from a single source. The second one was used to study the multi-source download, with the files being shared by more than one super-probe of the network.

Table 12 – Element's measurement group and correspondent host machine network card speed

Host Machine IP	Element Mode	Measurement Group ID	Network Card speed	
			Mbps	KB/s
193.136.92.107	Client	***a***	100 (fast)	12500
193.136.92.121	Super-probe	***a***	10 (slow)	1250
193.136.92.234	Super-probe	***b***	100 (fast)	12500
193.136.92.228	Super-probe	***c***	100 (fast)	12500
193.136.92.219	Super-probe	***d***	10 (slow)	1250

7.5.1 Single source download

In this section we describe the experiment performed to study the behavior of the single-source download process. To do so, we configured local searches on each super-probe of the network for 3 files with different sizes. For each super-probe and for each file, 10 downloads were configured. The average speeds of the 10 downloads are presented in Table 13 (the complete results are presented in the tables in appendix A.5).

Table 13 – Single Source Download Speed

File Size (KB)	Super-probe	Group ID	Speed (KB/s)	Delay (sec)
10	193.136.92.121 (slow)	***a***	291.327	0.035
	193.136.92.219 (slow)	***d***	78.052	0.129
	193.136.92.234 (fast)	***b***	256.109	0.040
	193.136.92.228 (fast)	***c***	246.785	0.045
500	193.136.92.121 (slow)	***a***	992.301	0.504
	193.136.92.219 (slow)	***d***	856.243	0.584
	193.136.92.234 (fast)	***b***	4903.229	0.104
	193.136.92.228 (fast)	***c***	5773.052	0.088
5120	193.136.92.121 (slow)	***a***	926.742	5.596
	193.136.92.219 (slow)	***d***	972.740	5.287
	193.136.92.234 (fast)	***b***	9068.546	0.565
	193.136.92.228 (fast)	***c***	9116.439	0.562

As can be verified in Table 13 and for files with larger sizes (500 and 5120 KB), the download speed from a given node approaches the maximum transfer rate the node supports (Table 12). For smaller files (10 KB), the download speed diverges from its real value. This may happen because smaller files are downloaded faster. Thus, the delay due to the exchange of the HTTP request and response messages and the processing delay will affect more the download speed.

As expected, the super-probes in the machines with IP addresses 193.136.92.121 (slow) and 193.136.92.219 (slow) have approximately the same download speeds. Also, the machines with IP addresses 193.136.92.234 (fast) and 193.136.92.228 (fast) have approximately the same download speeds, and these speeds are higher than the ones of the slower machines. This can be directly explained by the speeds of the network cards (Table 12). Note that the client element was running in the machine 193.136.92.107, which has a fast network card.

7.5.2 Multiple source download

As described in section 4.7.3, when the file to be downloaded is stored in more than one source, it can be downloaded simultaneously from multiple sources. In this section we present some experiments performed to test the multi-source download process. To do so, we configured downloads from 2, 3 and 4 sources for 3 different files. For the case of 2 sources two experiments were performed. In the first experiment, the files were stored at the machines 193.136.92.228 (fast – Group ID ***c***) and 193.136.92.219 (slow – Group ID ***d***). In the second experiment, the files were stored at the machines 193.136.92.121 (slow – Group ID ***a***) and 193.136.92.219 (slow – Group ID ***d***). The machines 193.136.92.228 (fast – Group ID ***c***), 193.136.92.219 (slow – Group ID ***d***) and 193.136.92.121 (slow – Group ID ***a***) were used in the 3 sources case. All the machines of the network in Figure 40 were used in the 4 sources case. For each case 10 downloads were performed and in Table 14 we present the average speeds and delays of these 10 downloads (the complete results are presented in the tables in appendix A.5).

Table 14 – Multiple Source Download Speed

File Size (KB)	Number of Download Sources	Group ID	Speed (KB/s)	Delay (sec)
500	2	c (fast) and d (slow)	6059.869	0.083
	3	a (slow), c (fast) and d (slow)	5798.623	0.087
	4	a, b, c, and d	4813.871	0.106
5120	2	c (fast) and d (slow)	7062.387	0.742
	2	a (slow) and d (slow)	1041.235	4.918
	3	a (slow), c (fast) and d (slow)	6257.745	0.838
	4	a, b, c, and d	5257.203	1.137
54133	2	c (fast) and d (slow)	6486.916	8.775
	2	a (slow) and d (slow)	1076.713	50.276
	3	a (slow), c (fast) and d (slow)	6668.692	8.268
	4	a, b, c, and d	6503.348	8.925

As can be verified in Table 14 for the 500 KB file case, the download speeds are close to the ones obtained with a single source (Table 13). This is because, as described in section 4.7.3, in a multi-source download process the requesting node should at least download 1

MB (minDownloadSize) from each remote node and when the file is smaller than 1 MB it should only be downloaded from a single source (the faster one).

For files bigger than 1 MB (5120 and 54133 KB in this case), the download speed depends on the number of download sources, on the speed of the network card of each source machine and on the download speed of the requesting element. As described in section 4.7.3, in these cases first the requesting element computes the maximum number of sources required to download at least say 1 MB (minDownloadSize) from each source. If the number of available sources is less than the required maximum, which is the case, all sources should be used. In these cases, the requesting element first divides the file content in equal sections to be downloaded from all available sources. Each section should be downloaded from different sources. In case one source finishes downloading, the requesting element should verify if there are more sections being downloaded. The section being downloaded by the slower source should be divided between the free source and the source currently downloading it, in a way that both sources lasts the same time to download their part of the section. This split is only performed if the remaining bytes to be downloaded are greater than 1 MB.

From now on we will refer to the download sources using the corresponding super-probe measurement group.

Table 14 one can verify that when there was only two download sources, one quite faster than the other one (***c*** (fast) and ***d*** (slow)), the files were downloaded faster than the case where only the slower source was used (***d*** (slow)) and slower than the case where only the faster source was used (***c*** (fast)) (Table 13 results). Note that the client was running in a machine supporting the same speed supported by the faster source (Table 12). Due to our method of multiple source download, the fragment of the file to be downloaded from the faster source will be larger than the one to be downloaded from the slower source. This is the reason why the download speed is closer to the download speed obtained when only the faster source was sharing the file.

When there are only two download sources with almost the same speed and which speed is smaller than the requesting client speed (***a*** (slow) and ***d*** (slow)), the file is downloaded faster than if only one of them was used (Table 14 and Table 13). This situation occurs because almost the same amount of the file content is downloaded simultaneously from both sources at the maximum speed data can be transferred from them.

As can be noticed in the results obtained when the file is downloaded from the measurement groups ***a*** (slow)**,** ***c*** (fast) and ***d*** (slow), the obtained download speed approaches the speed of the faster available download source (***c*** (fast)). In this case, the system has the same behavior described in the situation where the file was downloaded from a source quite faster than the other one (***c*** (fast) and ***d*** (slow)).

For the same reason described for the situations where the file is simultaneously downloaded from different sources and one of the sources is faster than the other ones (***c*** (fast) and ***d*** (slow); ***a*** (slow), ***c*** (fast) and ***d*** (slow)), in the situation where the file is downloaded from all the four existent nodes (***a***, ***b***, ***c***, and ***d***), the obtained results approaches the results obtained when only the faster source is used. However, the obtained speed is smaller than the speed of the faster sources because a section of the file is also

downloaded from the slower sources. In this case too, the obtained download seep is smaller than the speed achieved when only 3 nodes were sharing the files (***a*** (slow), ***c*** (fast) and ***d*** (slow)). This is verified because when there are four sources there will be more HTTP requests and responses messages used to request the required sections from each source and there will be more overhead associated with the download process. Thus, there will be larger delays.

7.6 Download speed comparison

In this section we describe some experiments performed to compare the download speed of the implemented system with the download speed obtained by other applications. The Internet Explorer (IE), FlashGet download manager and the LimeWire file sharing application were used to carry out this comparison.

7.6.1 IE and FlashGet

In this section we describe the experiments carried out to compare the download speeds obtained with the DTMS-P2P system with the ones obtained when using Internet Explorer (IE version 6) and FlashGet download manager (version 1.72). Two experiments were configured and to perform them copies of a file with 5120 KB and of a file with 54133 KB were placed in web servers installed in all machines of the network of Figure 40.

The first experiment was performed to compare the download speed obtained from each node using the DTMS-P2P system (single source download – results presented in section 7.5.1) with the ones obtained using IE and FlashGet applications. For each experiment, we performed 10 downloads from each node using the file with 5120 KB. The average speed of the 10 downloads are presented in Table 15 (the complete results are presented in the tables in the appendix A.5).

Table 15 – Single Source Download Speed Comparison

File Size (KB)	Web Server Host Machine IP	Speed (KB/s)	
		IE	FlashGet
5120	193.136.92.121 (slow)	981.4	870.397
	193.136.92.219 (slow)	994.7	938.665
	193.136.92.234 (fast)	5120	1280
	193.136.92.228 (fast)	5120	1280

The obtained results, presented in Table 15, indicate that the download speed obtained using IE is greater than the download speed obtained using FlashGet. For the slower machines, the results are only slightly better, but for the faster machines they are significantly better.

When comparing the results obtained with IE (Table 15) with those obtained with the DTMS-P2P system (Table 13), one verifies that, for the slower machines, the download speeds are approximately the same but, for the faster machines, the DTMS-P2P system performs much better, achieving almost twice the download speed of IE.

The second experiment was configured to compare the multi-source download process proposed in the DTMS-P2P system with the one implemented by the FlashGet download manager. To perform this experiment, the download manager was configured to

simultaneously download the same file from 2, 3 and 4 sources. For each case, the average speed of 10 downloads are presented in Table 16 (the complete results are presented in the tables in the appendix A.5).

Table 16 – Multiple Source Download Speed Comparison (FlashGet)

File Size (KB)	Number of Download Sources	Group ID	Speed (KB/s)
5120	2	c (fast) and d (slow)	1706.67
	2	a (slow) and d (slow)	1024
	3	a (slow), c (fast) and d (slow)	1621.336
	4	a, b, c, and d	1877.336
54133	2	a (slow) and d (slow)	998.876
	4	a, b, c, and d	2533.427

From the results in Table 16 it is possible to verify that, for the FlashGet download manager, both when (i) the file was stored in a faster and in a slower download source (***c*** (fast) and ***d*** (slow)) and when (ii) was only stored at the slower sources (***a*** (slow) and ***d*** (slow)), the obtained download speed is greater than the one obtained when only one of the sources is used (Table 15). This result from the fact that FlashGet is not able to download files from a single source at the maximum rate the FlashGet host machine can support (Table 15), which is not the case of the DTMS-P2P system as described in section 7.5.2. Thus when FlashGet uses multiple sources higher speeds are obtained in relation to the single source case, by exploring the speed left available at the FlashGet host machine.

In the multi-source download process, the FlashGet manager splits the file to be downloaded in equal sections to be downloaded from each source. When the requesting element ends downloading a section from a faster source, this source can be used to download half the remaining section of a slower source. Due to this behavior, when comparing the download speeds obtained for the two files used in these experiments (file with 5120 KB and file with 54133 KB) when they are stored in 4 sources, it is verified that a greater download speed is obtained for the larger file. This situation happens because the larger the file the larger will be the percentage of the file content that will be downloaded from the faster source(s). Thus, in these cases, the larger the files the greater will be their download speed.

When comparing the results of the multi-source download using the DTMS-P2P system (Table 14) and using FlashGet (Table 16), it is possible to verify that similar download speeds are obtained when using only the two slower download sources (***a*** (slow) and ***d*** (slow)). However, when using one of the faster sources, better results were obtained using DTMS-P2P implementation. This situation is verified due to the fact that the DTMS-P2P system try to download the files at the maximum speed supported by the involved elements and a greater section of a file is downloaded from the faster sources. FlashGet uses a different approach that will force the faster sources to download smaller sections of a file when compared with the proposed system. The approach used by FlashGet leads to smaller download speeds because it does not have in consideration the speed of the involved nodes (free source and slower source), when splitting the remaining bytes of a section being downloaded by a slower source.

7.6.2 LimeWire

In this section we describe the experiments performed to compare the download speeds of the DTMS-P2P system and of the LimeWire file sharing application (LimeWire Basic – version 4.12.11). To perform these experiments the LimeWire application was installed at the Windows machines of the network in Figure 40 and all the machines were interconnected. The LimeWire applications running at the machines that acted as super-probes in previous experiments were configured to share copies of a file with 54133 KB. The machine that hosted the client in previous experiments was used to download the file from the nodes sharing it using the LimeWire application.

In this case two experiments were configured. The first experiment was performed to compare the download speed obtained with the DTMS-P2P system (single source download – Table 13) with the one obtained with LimeWire. In the second experiment, LimeWire was used to download the file from multiple sources, with the purpose of comparing its performance with the one of the multi-source download approach proposed in this work. For each experiment 10 test runs were performed and the average values are presented in Table 17 (the complete results are presented in the tables in the appendix A.5).

Table 17 – Download Speed Comparison (LimeWire)

File Size (KB)	Number of Download Sources	Group ID	Speed (KB/s)
54133	1	b	2431.2
	3	b, c, and d	2662.1

When comparing the results obtained using the LimeWire application (Table 17) with those obtained with the DTMS-P2P system (Table 13 and Table 14) one notices that better results are achieved when using the proposed system, both when downloading from a single source or from multiple sources. In the case of multiple sources download, this behavior is verified because, as the in FlashGet application, the LimeWire application does not have in consideration from which source it should download a larger section of the file. For LimeWire (section 5.1) using HTTP 1.1 the application divides initially the file to be downloaded in equal regions (with a fixed small size) and starts the downloading the first regions from the available source. When a region is completely downloaded from a source, a pending region starts being downloaded from the same source and this process is repeated until all regions of the file are downloaded. Thus LimeWire thus not directly accounts for the download speeds of the sources, and spends a lot of time with the requests for downloads of the (small) regions, which is not incurred in the DTMS-P2P system.

7.7 Summary

In this chapter a set of experiments to evaluate and to validate the implemented DTMS-P2P system were performed. These experiments were performed to analyze the performance of the implemented system in terms of the time spent by an element to connect to an established system, to configure test sessions and to retrieve the results of test sessions.

A set of experiments to study the behavior of the system when downloading files from single and multiple sources were also performed. The results obtained are very satisfactory

when compared to the results obtained using other systems to process HTTP download (IE and FlashGet) or in comparison to the results obtained using the LimeWire file sharing application. In all cases, better results were obtained when using the proposed system.

Chapter 8. Conclusions

In this chapter we present the summary of this dissertation and we identify the main contributions of the present work. Moreover, we provide some guidelines for future work.

In section 8.1 we present the summary of this dissertation and we identify the main contributions of this work. Then, in section 8.2 we present the open questions and provide some guidelines for further research.

8.1 Summary and contributions

Currently there is a growing necessity to monitor Internet traffic. Traffic monitoring systems provide administrators with a tool to detect and respond to network events or behaviors that can have a significant impact on the network performance. There is a large variety of monitoring systems that can be used by network administrators, to study the behavior of the networks under their control [NMT].

Traditional traffic monitoring systems either rely on a single probe [TCPdump] [Ethereal] [NTOP] [MRTG] or in a centralized architecture where a set of probes are controlled by a single manager [RFC4656] [RFC3917] [RFC1757] [NMT]. The single probe system only monitors the traffic at one location and, therefore, is not flexible enough for medium and large size networks. The centralized architecture is able to provide an accurate view of the network status. However, it relies on a single manager, which makes it vulnerable to failures. Moreover, it stores measured data at a single collector, which may consume significant bandwidth when downloading data from probes.

To solve the problems related to the traditional traffic monitoring tools, some monitoring systems based on peer-to-peer (P2P) architectures have been proposed by some researchers. These type of monitoring systems have been proposed by Srinivasan and Zegura [Srinivasan2002] [Srinivasan2003]; by Liu, Boutaba and Hong [Liu2004] [Liu2005]; by Finkenzeller, Kunzmann, Kirstädter, Schollmeier [Finkenzeller2006]; and by Rabinovich, Triukose, Wen, and Wang [Rabinovich2006] [Wen2007] [DipZoom].

The three first P2P based monitoring systems mentioned above are equipped with built-in measurement modules used to perform active and passive measurements. This is disadvantageous because a user is not able to install and use other monitoring modules to perform measurement tests. On the contrary, the last system [DipZoom] may support any monitoring module but requires the development of plug-ins to support them. All these systems have rudimentary storage capabilities, which make it difficult to handle large data files and restrict the type of measurements that can be carried out. This is because the results of the test measurements are only stored at the nodes where the measurements were carried out and are not replicated at other nodes. Moreover, their architectures are not hierarchical and, therefore, do not scale well with the number of measuring nodes.

To overcome some limitations of the monitoring systems mentioned above, in this dissertation we propose a versatile, scalable and easily manageable traffic monitoring system based on a P2P hierarchical architecture (DTMS-P2P).

The DPMS-P2P system creates an overlay network to allow remote configuration of test sessions at the nodes of the network and the retrieval of the obtained results. The adoption of a P2P architecture allows high tolerance to failures and distributed storage of measured data. This architecture is also advantageous for traffic monitoring in wide area network environments. Moreover, access and querying of measured data can be performed using traditional P2P file sharing schemes. The system supports both IPv4 and IPv6 IP addresses and it allows the configuration of a set of parameters to guarantee its scalability.

DTMS-P2P can be used to perform both active and passive measurements since it supports any monitoring module that can be executed from command line, as long these modules are installed at the nodes of the network and these nodes are configured to permit their utilization. The results of the test sessions are stored at the nodes where they were produced and, to guarantee availability, may be replicated at other nodes of the system. The system uses traditional P2P file sharing schemas, to find and download these results (possibly from multiple sources using the new download mechanism proposed by us). Besides these functions, the system also allows network administrators to remotely request nodes of the network to process some other actions. For example, perform a file replication and/or deletion; provide the information about the available resources; connect to another node; provide the list of files the node is sharing; etc. Additionally, the system allows network administrators to obtain a full view of the network status under their control, by providing them the information about the existent elements and the round-trip time statistics between them.

The proposed system comprises some security features that should be used to prohibit theft of service and to provide secrecy of exchange. The system supports three security modes: unauthenticated, authenticated and encrypted. In the first security mode messages are sent in clear text. In the two other security modes the control messages are authenticated and encrypted. These two security modes only differ in the file transference process where the file blocks used to transfer the files are only encrypted in the encrypted security mode.

DTMS-P2P can be used as a large-scale measurement infrastructure in a community-oriented network where Internet users may share some processing power and storage space of their machines to allow other Internet users (researchers) to perform measurements, to retrieve and share the obtained results. Several community-oriented infrastructures are in common use among researchers. Some provide dedicated hardware (Skitter [Claffy1999], Ark [Ark], PlanetLab [Peterson2002] [PlanetLab], RON [Andersen2001]) and others implement an @home-style distributed measurement network (NETI@home [Simpson2004] [NETI@home] and DIMES [Shavitt2005] [DIMES]). In the @home-style approach it is provided a downloadable tool to be installed at the Internet users' home machines which allow them to share their resources to other users.

The proposed system can also work as an alarm system on which a node informs another node of a change in the network status, whenever it occurs.

In this dissertation we also compared the similarities and differences of the file search and download mechanisms of the proposed system with the ones used by the LimeWire file sharing application and the ones used by applications implementing the BitTorrent protocol. The file search mechanism implemented in LimeWire, which is based on the Gnutella protocol, is similar to the one used in DTMS-P2P. The file search mechanism is

not necessary in applications implementing the traditional BitTorrent protocol since the information about the files shared in the network are provided using the “torrent” file. The proposed download mechanism is similar to the one used in the LimeWire application, but it has some improvements which allows faster downloads. This mechanism is completely different from the one used in the BitTorrent protocol.

We also compared DTMS-P2P with the P2P systems based in the Distributed Hash Table (DHT) method. These last ones implement a completely different approach to build the overlay network, to share, retrieve and replicate files. In particular they are not well adapted to networks where the nodes may have insufficient storage capacity.

In order to create a platform that can represent a basis for the development and test of the proposed system, an implementation of the system was built. The DTMS-P2P system was developed in Java language because this language presents a set of favorable characteristics, like semantic simplicity, portability and a set of classes that greatly simplify the construction of distributed applications.

A set of tests to validate the implemented DTMS-P2P system were performed. These tests evaluated the performance of the implemented system in terms of the time spent by an element to connect to an established system, to configure test sessions and to retrieve the results of test session.

By the analysis of the obtained results, one may conclude that the implemented system has a good performance and is a good platform to assist network administrator in the configuration of tests session at the elements of the network under their control.

8.2 Open questions and further research

In this work we proposed the DTMS-P2P system and we defined a set of functionalities the system should support. In this section we will suggest some new functionalities that should be analyzed to be included in the future.

A new functionality that may be added to the proposed system is the possibility to remotely install a given monitoring module at a node connected to the network. This may be advantageous in the sense that a user may use a client to request a remote node to download and install a given monitoring module he wants to use. Then, the user will be able to configure test sessions at the selected node, using the respective monitoring module.

To prevent the overloading of the system elements, they should be configured to only maintain a given number of node addresses at their Cache of Known Nodes, and they should be configured to only establish a given number of connections when in super-probe mode. Due to this behavior, it may be possible that not all the nodes of the system are interconnected. To prevent this situation, an algorithm to guarantee the connectivity between all the elements of the system should be researched.

The proposed method to guarantee the security of the system only provides secrecy of message exchange. A method to guarantee controlled user access to the nodes can be proposed in the future. One may define different users and different groups of users. For

each group of users one may define different levels of permissions. For example, permission to configure test sessions, permission to request the download of test results, permission to install new monitoring modules, etc.

At the moment the proposed system only allows the configuration of test sessions and results retrieval. A new functionality that may be included is the possibility to incorporate statistical analysis tools at the nodes of the system. These tools may be used to compute the required statistical data over the results of configured test sessions. A user may only retrieve the statistical data instead of the large files with the obtained results. This is very advantageous because the user can obtain the statistical data directly from a node without needing to process the test session results at the client side, thus reducing the required processing capabilities of the equipments running a client element.

The length of the Heavy Data Files may be very large and the statistical analysis on their content may be an expensive process. Thus, it may not be advantageous to replicate these files or to download them to process their content at the client side. Using the procedure described above, nodes may compute and only replicate the files with the statistical data obtained from the Heavy Data Files they produce. The files with statistical information are smaller than the files with the results of a test session. Thus, less storage space will be occupied at the nodes where they are replicated, faster downloads will be achieved and less bandwidth will be occupied during data transfers.

The system may also allow a user to request a given node to download a result's file from another element (its client or another node) and request it to compute the statistical data over the results in the respective file. This is very advantageous because a user may request a node, running in a machine with more processing capacity than the node where a Heavy Data File was produced or the client he is using, to download a Heavy Data File and then request it to process the results of the configured test session.

All the processes described above are very advantageous because for example, a user may run a client in a device with low processing capabilities, such as a mobile phone, and request test sessions at any node of the system. Then, the user can only retrieve the statistical information computed over the obtained results.

Additionally, a data compression module can be included in the nodes of the DTMS-P2P system to be used to compress the Heavy Data Files before replication or download. Compressing these files will result in smaller files which will be downloaded faster and which transference will occupy less network bandwidth. At the client side these files can then be decompressed to be read.

Another useful functionality that may be included is the possibility to get statistical information about how many files a node is sharing, how many bytes it is sharing, how many measurements it performed, etc. This information can be used to build the history of all configured measurements.

In the proposed system the nodes consider that they are firewalled or not as configured by the node's local administrator. However, it may be possible that the node's administrator does not know if the node is or not firewalled. Thus, it may be useful to capacitate the nodes to determine on their own if they are or not firewalled right after the connection set-

up. To do so, a node may request a remote node to open a connection to it, to be able to determine if it can or not accept connection requests.

Another functionality that may be included to the proposed system is a module to draw the network topology based on the information of the Light Data File.

A mechanism to allow the system to work as an alarm system on which nodes may inform other nodes of a change in the network status, whenever it occurs, should also be matter of study.

In the proposed replication mechanism the number of replicas existent in the network decreases if a node with file replicas leaves the system. A mechanism to try to maintain constant the number of replicas in the network should be studied in the future.

Finally, another functionality to be implemented in the future is the possibility to request a given node to replicate a file at another specific node. Using this functionality a user may request a node behind a firewall to replicate a given file at a not firewalled location, from where the user can download the required file.

Appendix

A.1 Measurement group identification (Group ID)

In the DTMS-P2P protocol the Group ID of the measurement groups must be represented by a sequence of 16 pairs of hexadecimal characters uniquely identifying the measurement group at the network. An element's measurement group ID must be provided by its administrator and indicates to which measurement group the element belongs. The hexadecimal sequence to be provided should have at least a pair of hexadecimal characters and at maximum 16 pairs of hexadecimal characters. If the provided sequence has less than 16 pairs of hexadecimal characters, the sequence must be completed by adding the necessary number of 00 hexadecimal characters to the beginning of the sequence. A Group ID must not have the initial 0x character commonly used when representing hexadecimal characters. Each hexadecimal character should be written using the symbols 0-9 and A-F, or a-f. By default an element's Group ID is equal to "00000000000000000000000000000000". The Group ID can also be represented as any sequence with a maximum of 16 characters which will then be converted to a sequence of 16 pairs of hexadecimal characters to be used in the protocol.

A.2 Format of the messages of the DTMS-P2P protocol

All multi-octet quantities defined in this document are represented as unsigned integers in network byte order unless specified otherwise.

A.2.1 Message Header

As described in Chapter 4, all elements of the network communicate with each other by exchanging a set of control messages. Not all the messages have a fixed size, but all messages (except the three first messages used in connection set-up process – section 4.3.5) have a common header with 40 bytes. The message header is divided into the following fields:

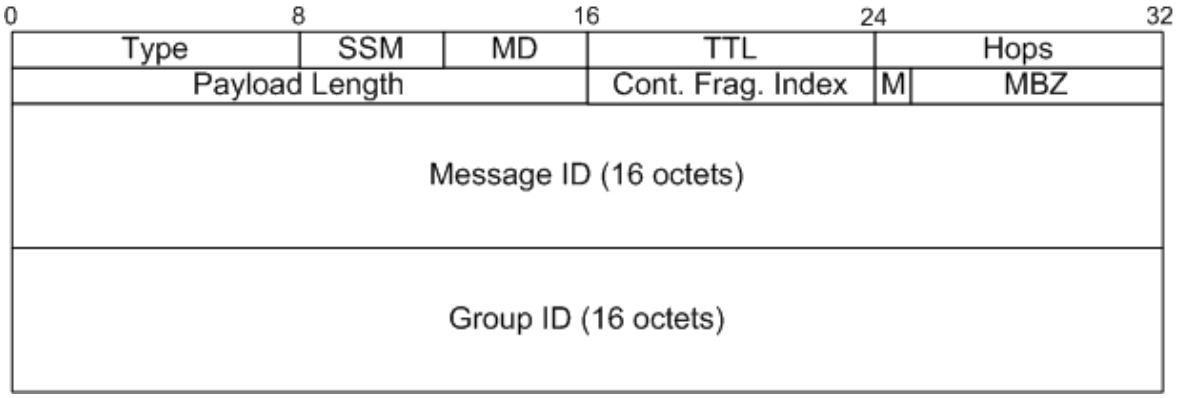

Figure 41. Message header.

The first byte of the message header is used to define the type of the message:

0x00 Ping

0x01 Pong

0x02 List Of Supported Monitoring Modules

0x03 List Of Shared Files

0x04 Demotion Negotiation

0x05 Measurement Group Discovery Request

0x06 Measurement Group Discovery Response

0x07 List of Nodes Discovery Request

0x08 List of Nodes Discovery Response

0x09 List of Supported Monitoring Modules Request

0x0A List of Supported Monitoring Modules Response

0x0B Monitoring Module Help Request

0x0C Monitoring Module Help Response

0x0D Monitoring Module List of Restrictions Request

0x0E Monitoring Module List of Restrictions Response

0x0F Command

0x10 Command Response

0x11 Potential Storing Nodes Discovery Request

0x12 Potential Storing Nodes Discovery Response

0x13 Replication Request

0x14 Replication-Ack

0x15 Download Replication-Ack

0x16 Push

0x17 Query

0x18 Query-Hit

0x19 File Action Request

0x1A File Action Response

0x1B Resources Request

0x1C Resources

0x1D Light Data

0x1E Connect To Node Request

0x1F Connect To Node Response

0x20 Bye

The second byte of the message header is divided in two sets of 4 bits: SSM (Supported Security Modes) and MD (mode). The SSM field is used to identify the security modes the sending element supports. The following modes values are meaningful: 1 for unauthenticated, 2 for authenticated and 4 for encrypted. The value of the SSM field sent is the bit-wise OR of the security mode values that the sending element is willing to support. Thus, the last three bits of the SSM 4-bit value are used. The first bit must be zero. The receiving element must ignore the values in the first bit of the SSM value. This way, this bit is available for future protocol extensions. This is the only intended extension mechanism. Therefore if an element is configured to support the three security modes, the last three bits of the SSM field must be set (equal to 1). The MD field is used to code the type of the system element that is sending the message: 0 for probe, 1 for super-probe and 2 for client.

The TTL (Time To Live) field is the number of times the message will be forwarded by super-probes before it is removed from the network. Each super-probe will decrement the TTL before passing it on to another super-probe. When the TTL reaches 0, the message will no longer be forwarded (and must not).

The Hops field is the number of times the message has been forwarded. As a message is passed from super-probe to super-probe, the TTL and Hops fields of the header must satisfy the following condition:

$$TTL(0) = TTL(i) + Hops(i)$$

Where TTL(i) and Hops(i) are the value of the TTL and Hops fields of the message, and TTL(0) is maximum number of hops a message will travel (usually 7).

The next two bytes represent the message Payload Length. It is the payload length in bytes of the message immediately following this header. The next message header is located exactly this number of bytes from the end of this header i.e. there are no gaps or pad bytes in the DTMS-P2P data stream. The Payload Length field is the only reliable way for a node to find the beginning of the next message in the input stream. Therefore, elements should rigorously validate the Payload Length field for each message received. If an element becomes out of synch with its input stream, it should close the connection associated with the stream since the upstream element is either generating or forwarding invalid messages.

Messages should not be larger than 4 KB. If the information to be sent in a message (message payload) will force the message to exceed this size, it must be fragmented to be sent in messages not larger than 4 KB. The sending element should send these messages, one by one, until the overall information to be sent is completely dispatched. The message's header Cont. Frag. Index (Content Fragment Index) and M (More Fragments) fields are used to verify if the content to be received was subdivided in more than one fragment. The Cont. Frag. Index field is used to store the index of the fragment being sent in the message. Its value must be between zero (first fragment) and number of fragments less 1 (last fragment). The M field must be set to 1 if the content of the message is not the last fragment. When set to zero, the message contains the last fragment. A destination element receiving a message with this field set to one must wait to receive all the fragments before using the received information. Transit super-probes are not required to wait to receive all the fragments before forwarding the received messages. After receiving the last fragment (M field set to zero) the destination element is able to compute the number of fragments to be received which will be equal to the Cont. Frag. Index field of the message with the last fragment plus one. Thus, using these two fields an element is able to determine the number of fragments to be received and then use the Cont. Frag. Index fields of each received message to rebuild the original content information sent by the sending element.

When the information to be sent is subdivided in different fragments to be sent in different messages, all these messages must have the same Message ID. Moreover, whenever possible, the content of the information to be sent must be divided in such a way that one fragment should not depend on the other fragments. Thus, if the content to be sent is a string, fragments should be of maximum size, whenever possible. Otherwise, a fragment must only contains the maximum number of objects (e.g. IP address, hash codes, etc) that will not make the message exceeds 4 KB.

The MBZ (Must Be Zero) field must be set to zero. Here and hereafter this field have the same semantics: the party that sends the message must set the field to a string of zero bits and the party that receives the message must ignore it. This way this field could be used for future extensions.

The Message ID field (16 bytes) is a string uniquely identifying the message in the network. It should be constructed by concatenation of the 4-octet IPv4 IP number belonging to the generating machine, an 8-octet timestamp, and a 4-octet random value. To reduce the probability of collisions, if the generating machine has any IPv4 addresses (with the exception of loopback), one of them should be used for Message ID generation, even if all communications are IPv6-based. If it has no IPv4 addresses at all, the last four octets of an IPv6 address may be used instead. If truly random values are not available, it is important that the Message ID be made unpredictable.

The last 16 bytes of the message's header (Group ID field) are used to identify the measurement group of the element where the message was generated (appendix A.1). The element's measurement group ID is a 16 byte string uniquely identifying the group in the network. To build the message's header Group ID field, each pair of hexadecimal character of the element's Group ID should be converted to a byte of the message's header Group ID field. This field should not be changed by the nodes that are flooding a received message.

A.2.2 Server Greeting

The Server Greeting message comprises the following fields:

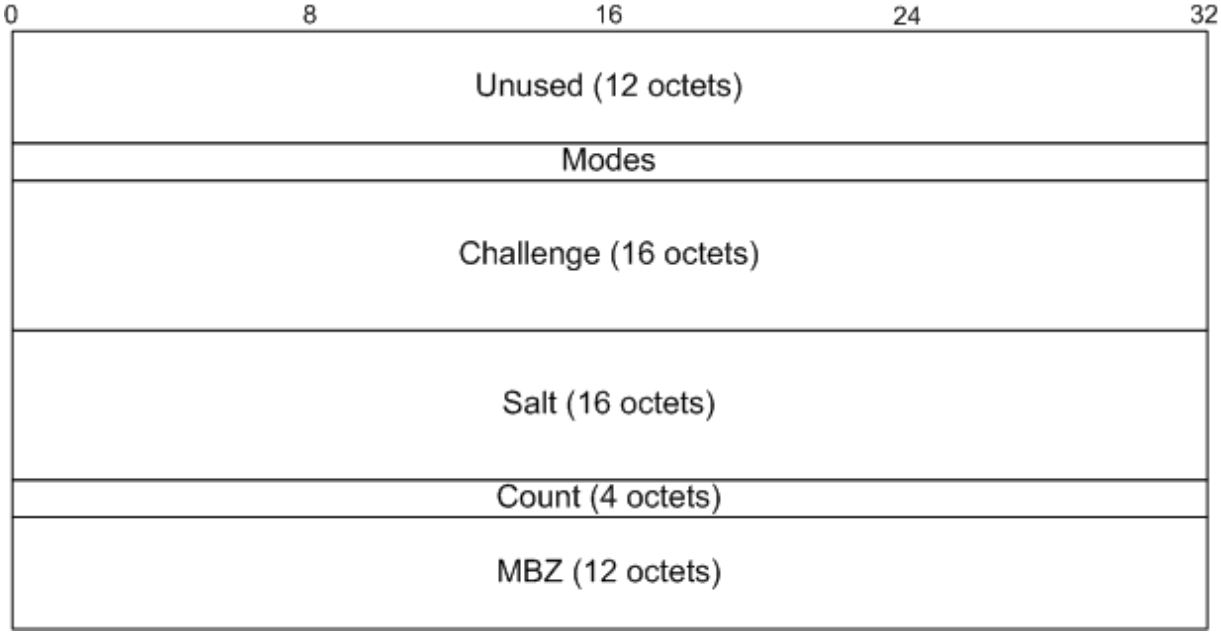

Figure 42. Server Greeting.

The first 12 octets of this message should be ignored.

The Modes field indicates the modes of security the sending element supports. The following Modes values are meaningful: 1 for unauthenticated, 2 for authenticated and 4 for encrypted. The value of the Modes field sent is the bit-wise OR of the security mode values that the sending node is willing to support in the requested connection. Thus, the last three bits of the Modes 32-bit value are used. The first 29 bits must be zero. The requesting element must ignore the values in the first 29 bits of the Modes value. This way, the bits are available for future protocol extensions. Therefore if an element is configured to support the three security modes, the last three bits of the Modes field must be set (equal to 1). If the Modes value is zero, the responding element does not wish to communicate with the requesting element and may close the connection immediately. The requesting element should close the connection if it receives a greeting with Modes equal to zero. The requesting element may also close the connection if its desired mode is unavailable.

The Challenge field is a random sequence of octets generated by the sending element; it is used subsequently by the requesting element to prove possession of a shared secret in a manner prescribed in appendix A.2.3.

The Salt and Count fields are parameters used in deriving a key from a shared secret using a password-based key derivation function PBKDF2 (PKCS #5) [RFC2898], as described in section 4.1.2. Salt must be generated pseudo-randomly. Count must be a power of 2. Count must be at least 1024. Count should be increased as more computing power becomes common.

A.2.3 Setup Response

The Setup Response message comprises the following fields:

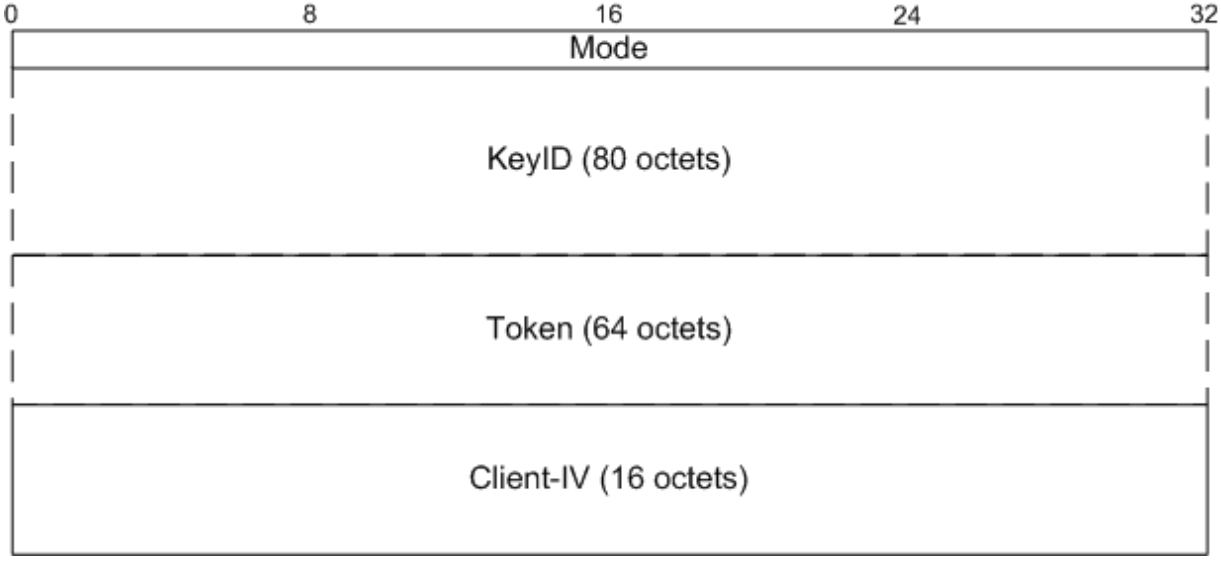

Figure 43. Setup Response.

The Mode field of the Setup Response message is the security mode that the requesting element chooses to be used in the new connection. In Mode, one or zero bits must be set within last three bits. If it is one bit that is set within the last three bits, this bit must indicate a security mode that the remote element agreed to use (i.e., the same bit must have been set by the remote element in the Server Greeting message). The first 29 bits of Mode must be zero. The remote element must ignore the values of the first 29 bits. If zero Mode bits are set by the requesting element, it indicates that it will not continue with the connection request; in this case, the requesting element and the remote element should close the TCP connection associated with the request.

In unauthenticated security mode, KeyID, Token, and Client-IV are unused. Otherwise, KeyID is a UTF-8 string, up to 80 octets in length (if the string is shorter, it is padded with zero octets), that tells the remote element which shared secret the requesting element wishes to use to authenticate or encrypt; while Token is the concatenation of the 16-octet challenge received in the Server Greeting message, a 16-octet AES key used for encryption, and a 32-octet HMAC-SHA1 key used for authentication. The token itself is encrypted using the AES (Advanced Encryption Standard) [AES] in Cipher Block Chaining (CBC). Encryption must be performed using an Initialization Vector (IV) of zero and a key derived from the shared secret associated with KeyID. (Both the remote element and the requesting element use the same mappings from KeyIDs to shared secrets. The remote element, being prepared to connect to more than one element, uses KeyIDs to choose the appropriate secret key; an element would typically have different secret keys for different elements. The situation is analogous to that with passwords.)

The shared secret is a passphrase; it must not contain new lines. The secret key is derived from the passphrase using a password-based key derivation function PBKDF2 (PKCS #5) [RFC2898] as described in section 4.1.2. The PBKDF2 function requires several

parameters: the PRF (underlying pseudorandom function) is HMAC-SHA1 [RFC2104]; the salt and count are as transmitted by the responding element (Server Greeting message).

AES Session-key, HMAC Session-key and Client-IV are generated randomly by the requesting element. AES Session-key and HMAC Session-key must be generated with sufficient entropy not to reduce the security of the underlying cipher [RFC4086]. Client-IV merely needs to be unique (i.e., it must never be repeated for different connections using the same secret key; a simple way to achieve that without the use of cumbersome state is to generate the Client-IV values using a cryptographically secure pseudo-random number source: if this is done, the first repetition is unlikely to occur before 2^{64} connections with the same secret key are conducted).

A.2.4 Server Start

The Server-Start message comprises the following fields:

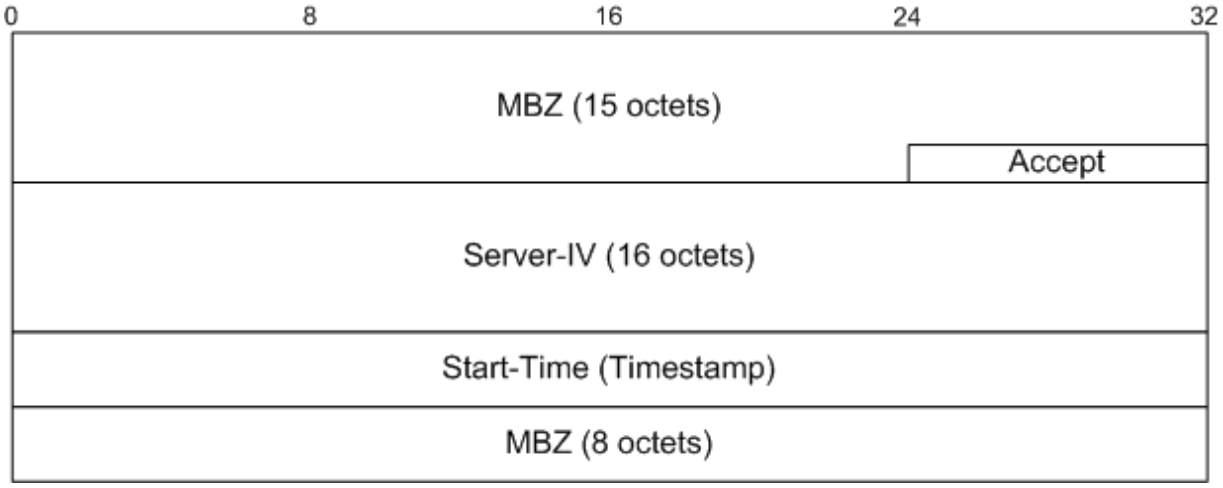

Figure 44. Server Start.

The Accept field indicates the sending element's willingness to continue communication. A zero value in the Accept field means that the sending element accepts the authentication and is willing to conduct further transactions. Non-zero values indicate that the sending element does not accept the authentication or, for some other reason, is not willing to conduct further transactions in this connection.

If a negative (non-zero) response is sent, the sending element may (and the requesting element should) close the connection after this message.

Server-IV is generated randomly by the sending element. In unauthenticated security mode, Server-IV is unused.

Start-Time is a timestamp representing the time when the current instantiation of the sending element started operating. (For example, in a multi-user general purpose operating system, it could be the time when the element process was started.) If Accept is non-zero, Start-Time should be set so that all of its bits are zeros. In authenticated and encrypted modes, Start-Time is encrypted as described in section 4.1.1, unless Accept is non-zero.

The format of the timestamp is the same as in [RFC1305] and is as follows: the first 32 bits represent the unsigned integer number of seconds elapsed since 0h on 1 January 1900; the next 32 bits represent the fractional part of a second that has elapsed since then.

The same instantiation of the node should report the same exact Start-Time value to each requesting element in each connection request.

A.2.5 Ping

The Ping message comprises the following fields:

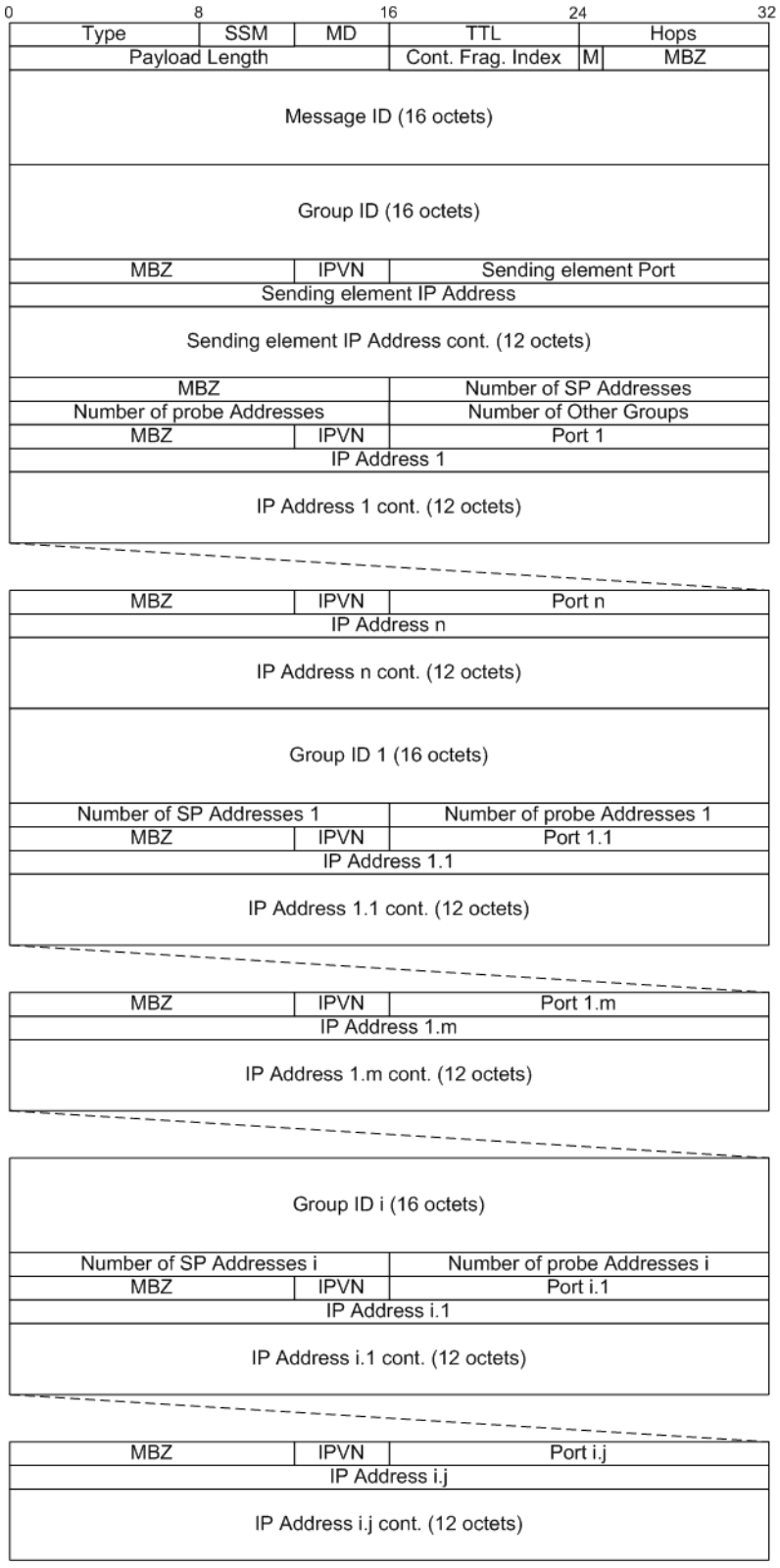

Figure 45. Ping.

The Sending element IP Address and Port fields represent the IP address and port number where the sending element is listening for new connections. IPVN is the IP version number to be used in the IP Address field. When the IP version number is 4 a 4-octet IPv4 address is stored in the IP Address field and the 12-octets IP Address cont. field is not sent. In the case the IP version number is 6 (IPv6), all the 16 octets of the IP Address field (IP Address + IP Address cont.) should be filled with the IPv6 element address. Currently meaningful IPVN values are 4 and 6. The IP Address and IPVN fields here and hereafter have the same semantics.

If possible, this message should have a list of addresses and port numbers of all super-probes and probes of the sending element's measurement group and a list of addresses and port numbers of all known nodes (super-probes and probes) of other measurement groups. These addresses are the addresses of the nodes the sending element is sure or believes that they exist in the network. This list represents the element's current list of known nodes in the network (this information is stored in the element's Cache of Known Nodes (CKN) – section 4.3.2). This list should not comprise the address of the receiving element neither the address of the sending element. Notwithstanding the fact that a node address is in this list, it may be possible that this node doesn't exist in the network. This may happen because the sending element didn't try a connection to that node yet.

The Number of SP Addresses is the number of super-probe addresses of the sending element's measurement group that will follow in the list of node addresses. The Number of probe Addresses is the number of probe addresses of the sending element's measurement group that will follow in the list of node addresses. The Number of Other Groups is the number of sub lists of known node addresses of nodes of other measurement groups that will follow in the list of node addresses. These sub lists comprise the Group ID of the respective measurement group (Group ID *i*), the number of super-probe addresses (Number of SP Addresses *i*) and the number of probe addresses (Number of probe Addresses *i*) that will follow, for the respective sub list. These node addresses are the known nodes of the corresponding measurement group. These sub lists must be placed sequentially after the list of known nodes of the sending element's measurement group.

All IP addresses and port numbers should be presented in the following order: super-probe addresses first then probe addresses.

For all node addresses in the list of node addresses:

- The IPVN field has the same meaning as explained before;
- The IP Address and Port fields have the IP address and port number where the probe/super-probe is waiting for incoming connections.

The list of node addresses is always sent in the Ping message. Since messages must not be larger than 4 KB (appendix A.2.1) the content of an element CKN must be fragmented to be sent in different Ping messages when it cannot be sent in only one message. When this happens, each Ping message must only contain the maximum number of address that will not make it exceeds 4 KB. Moreover, the information related to another measurement group should only be sent in a message if at least one address of that measurement group is going to be sent.

A.2.6 Pong

The Pong message comprises the following fields:

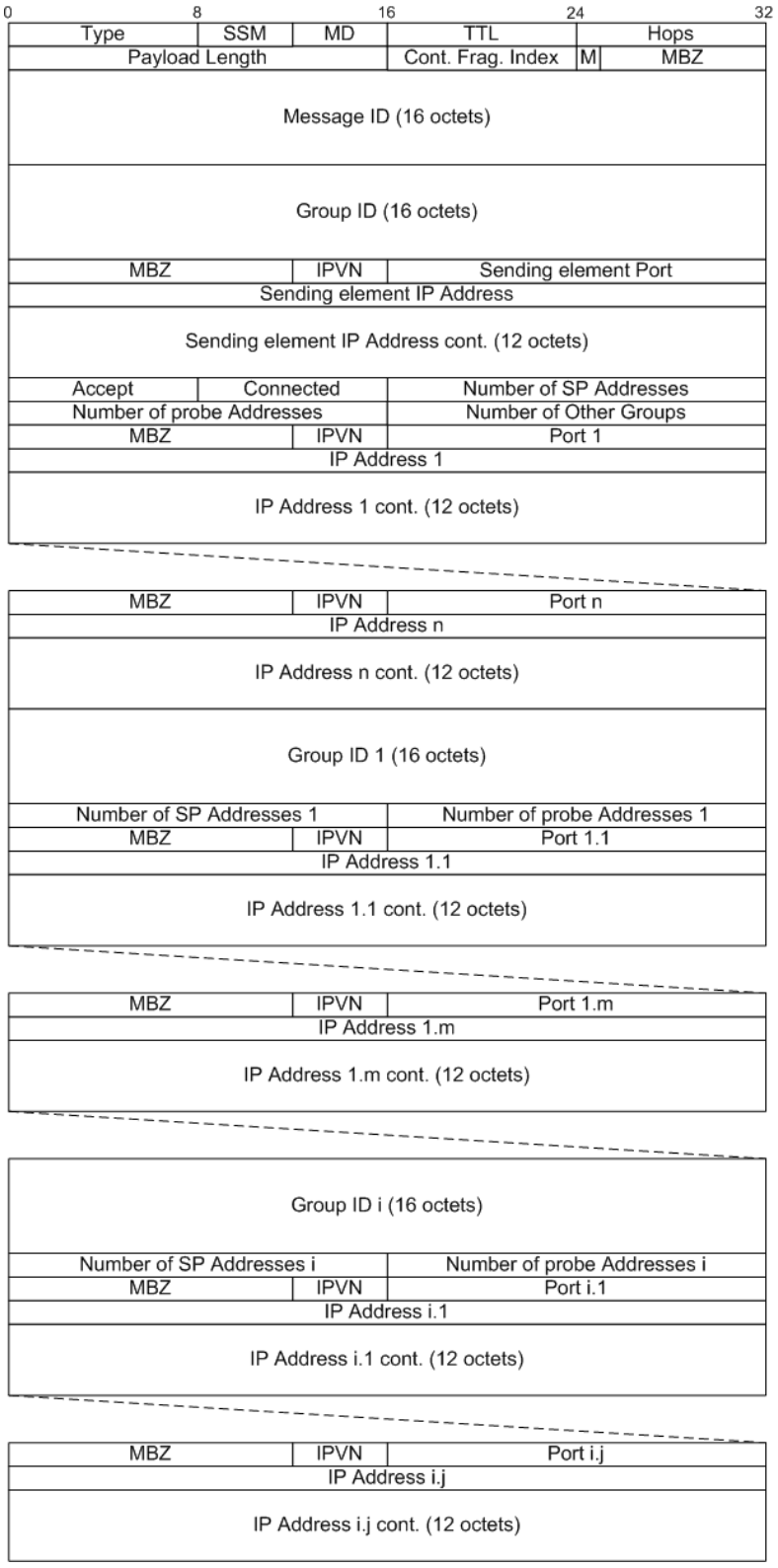

Figure 46. Pong.

The Message ID field of the Pong message must be equal to the Message ID of the Ping message it is sent in reply to in the connection set-up process (section 4.3.5). In the Pong flooding process (section 4.3.6.1.1) the Pong Message ID must be computed as described in appendix A.2.1.

The Sending element IP Address and Port fields represent the IP address and port number where the sending element is listening for new connections.

The Accept field indicates if the element (super-probe or probe) accepts or not the connection request. A zero value in the Accept field means that the element is willing to conduct further transactions. Non-zero values indicate that the element does not accept the connection request.

The element sending the request should disconnect if receiving any response with a non-zero value in the Accept field.

When the Accept field has a non-zero value it represents the code of the reason the sending element is rejecting the connection request. The following codes values are meaningful:

1 Failure, reason unspecified;

2 Internal error;

3 The responding element is already connected to the requesting element. This code is sent in case the responding element detects that a previous connection was already made to the requesting element. Thus, after receiving a Ping message the receiving element must always verify if a previous connection with the requesting element was already made. If yes, the connection request must be rejected;

4 The mean CPU usage is above a threshold;

5 The mean memory usage is above a threshold;

6 The free storage space is bellow a threshold;

7 The maximum number of connections allowed for this super-probe has been reached;

8 The maximum number of connections to probes allowed for this super-probe has been reached;

9 The network where the element is located is congested;

10 The responding element is a probe. This code is sent in the Pong message when a probe receives a Ping from a client or another probe or from a super-probe. This can happen in system startup or when the system is trying to recover from a failure;

11 The super-probe belongs to another measurement group. This code is sent in the Pong message when a super-probe receives a connection request from a probe/client of another measurement group;

12 Invalid Ping message. The Ping message received has the element's address or the address of the sending element in its list of known node addresses;

13 This code should not be used in the handshaking process. It is used when a super-probe is sending the Pong message, with its list of known nodes, to be flooded to all elements connected to the network in the process called Pong flooding (section 4.3.6.1.1).

As for the Ping message, the Pong message should comprise the list of node addresses the sending element believes exist at the network (this information is stored in the element's Cache of Known Nodes – section 4.3.2). This list should not comprise the address of the receiving element neither the address of the sending element. Also, it should not have the new addresses received in the Ping message. Thus, it should comprise only the known node addresses, before the Ping message was received. The Number of SP Addresses, the Number of probe Addresses, the Number of Other Groups and all the fields associated with the list of known node addresses have the same meaning as in the Ping message. Also, the message must not exceed 4 KB and, when required, the content of the element's CKN must be fragmented to be sent in different Pong messages, as in Ping message.

The Connected field is used to inform the receiving element if the sending element is or not connected to the network. If equal to zero the element is connected to the network.

A.2.7 List of Supported Monitoring Modules

The List of Supported Monitoring Modules message comprises the following fields:

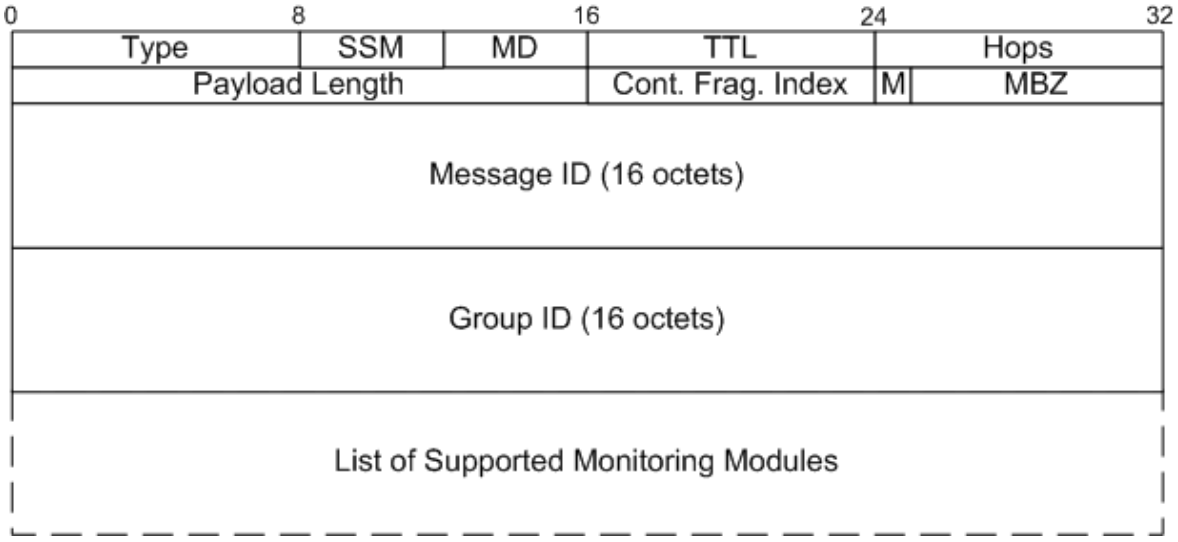

Figure 47. List of Supported Monitoring Modules.

The List of Supported Monitoring Modules field of this message represents the list of hash codes (appendix A.3) of the names of the monitoring modules the sending probe supports (converted to lower case). Using the hash codes of the monitoring module's names will generate smaller messages. The hash codes of the monitoring module's names are also used because the memory of the destination super-probe element will be used more efficiently since each word hash code only occupies 4 bytes (the name of the monitoring

modules may occupy more than 4 bytes). Remember that the receiving super-probe must store the received information to be able to determine which probes connected to it support a given monitoring module (section 4.4.1.2). As each hash code occupies only 4 bytes, with the message payload length it is possible to determine the number of hash codes in the List of Supported Monitoring Modules field.

A.2.8 List of Shared Files

The List of Shared Files message comprises the following fields:

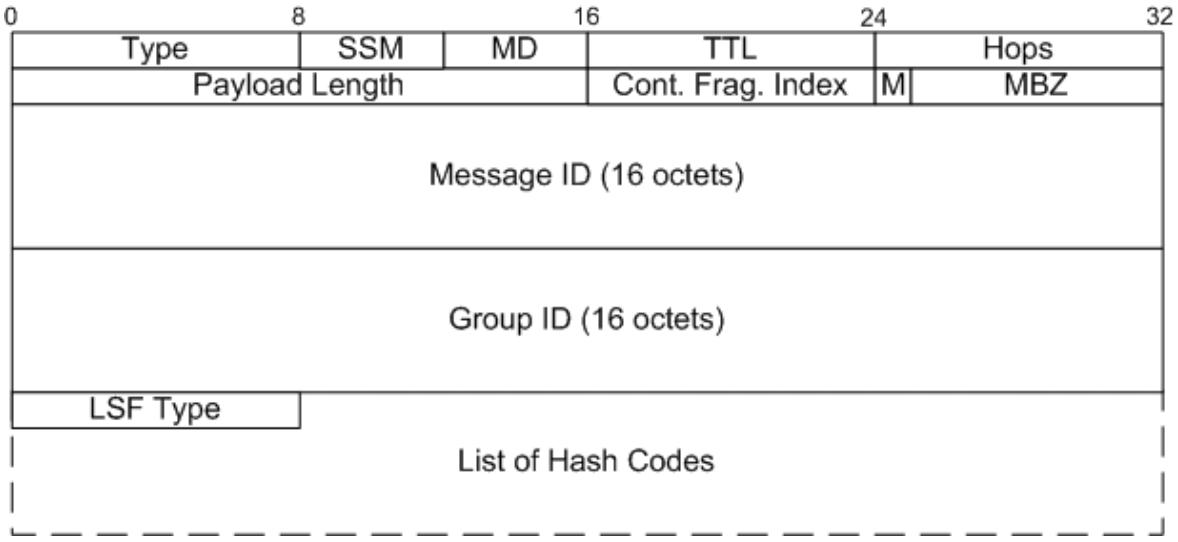

Figure 48. List of Shared Files.

The List of Shared Files message is sent in three different situations. It may be sent by a probe to inform its super-probe which files it is sharing (section 4.6.1). In this case the List of Hash Codes (appendix A.3) of this message has the hash codes of all words (all lower case) found in the names of the files the probe is sharing. The message is also used by a probe to send the information of new shared files to its super-probe (sections 4.4.3 and 4.5.3.2.2). In this last case the List of Hash Codes of this message only has the hash codes of all words (all lower case) found in the names of the new files the element is sharing. And the message is also sent by a probe to inform its super-probe that it is not sharing anymore files which names has the words which hash codes are sent in the message's List of Hash Codes (section 4.9.1). This last situation may happen when the probe is configured to delete a given Heavy Data File it was sharing. In this case, it may be possible that the probe is not sharing anymore any more files which names has one of the words present in the name of the deleted file. Thus, the probe must request its super-probe to remove these hash codes from the list of hash codes related to the files the probe is sharing. The List of Shared Files message's LSF Type (List of Shared Files Type) field is used to identify the type of this message. In the first situation the LSF Type field should be set to zero, in the second situation it should be set to 1 and in the third situation it should be set to 2.

To compute the hash codes of the words present in the name of a file, these words are obtained by breaking up the file name on any non-alphanumeric characters (anything but letters and numbers). A space is the standard separator between words. But the separator between words may also be the following characters: "_" and "-". Each hash code occupies

4 bytes. Thus, with the message payload length is possible to determine the number of hash codes of the message's List of Hash Codes field.

A.2.9 Demotion Negotiation

The Demotion Negotiation message comprises the following fields:

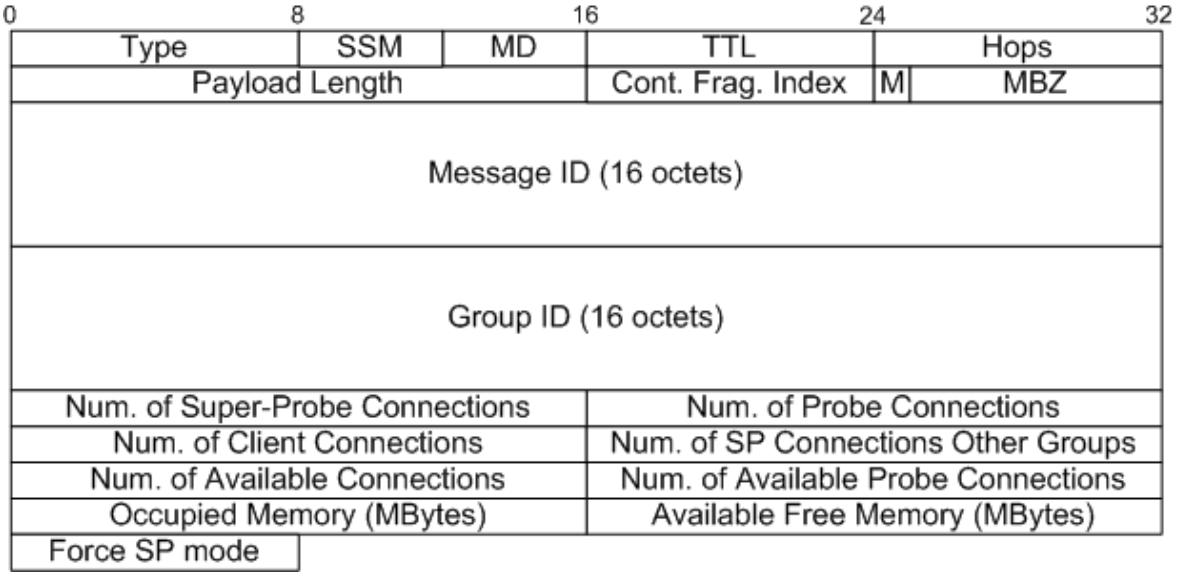

Figure 49. Demotion Negotiation.

In the demotion negotiation process (section 4.3.6.2), the Demotion Negotiation message sent by the responding super-probe must have the same Message ID of the Demotion Negotiation message sent by another super-probe to start the process.

The Num. of Super-Probe Connections field represents the number of connections to super-probes of its measurement group the sending super-probe currently has active.

The Num. of Probe Connections field is the number of connections to probes that the sending super-probe currently has active.

The Num. of Client Connections field is the number of connections to clients that the sending super-probe currently has active.

The Num. of SP Connections Other Groups field represents the number of connections to super-probes of other measurement groups the sending super-probe currently has active.

The Num. of Available Connections field represents the current number of connections the sending super-probe can accept from the elements connected to the remote super-probe. A super-probe can be configured to only make a given maximum amount of connections. During the demotion negotiation process, a super-probe must reserve an amount of the currently available connections to maintain free. This amount must be equal to the configured percentage (as set by the element's administrator) of the maximum number of connections the super-probe is configured to make. Thus, the Num. of Available Connections field must be equal to the current available number of connections the super-

probe can accept less the amount of connections to maintain free. If the obtained value is less than zero, the Num. of Available Connections field must be equal to zero.

The Num. of Available Probe Connections field represents the current number of probe connections the sending super-probe can accept from the probes connected to the remote super-probe. A super-probe can be configured to only make a given amount of probe connections. During the demotion negotiation process, a super-probe must reserve an amount of the currently available probe connections to maintain free. This amount must be equal to the configured percentage (as set by the element's administrator) of the maximum number of probe connections the super-probe is configured to make. Thus, the Num. of Available Probe Connections field must be equal to the current available number of probe connections the super-probe can accept less the amount of connections to maintain free. If the obtained value is less than zero, the Num. of Available Probe Connections field must be equal to zero.

The Occupied Memory field represents the amount of memory the super-probe's process is currently occupying (measured in MBytes).

The Available Free Memory field represents the current amount of available free memory, measured in MBytes, the sending super-probe can use to store the information about the elements connected to the remote super-probe that will connect to it in case the remote one is demoted to a probe. During the demotion negotiation process, a super-probe must reserve an amount of the currently available free memory to maintain free. This amount must be equal to the configured percentage (as set by the element's administrator) of the maximum amount of memory the super-probe process can use. Thus, the Available Free Memory field must be equal to the current available memory the super-probe can use less the amount of free memory to maintain free. If the obtained value is less than zero, the Available Free Memory field must be equal to zero.

Case the sending super-probe is configured to maintain the super-probe mode, the Force SP mode message's field should be equal to 0. Otherwise, it must be equal to 1.

A.2.10 Measurement Group Discovery Request

The Measurement Group Discovery Request message comprises the following fields:

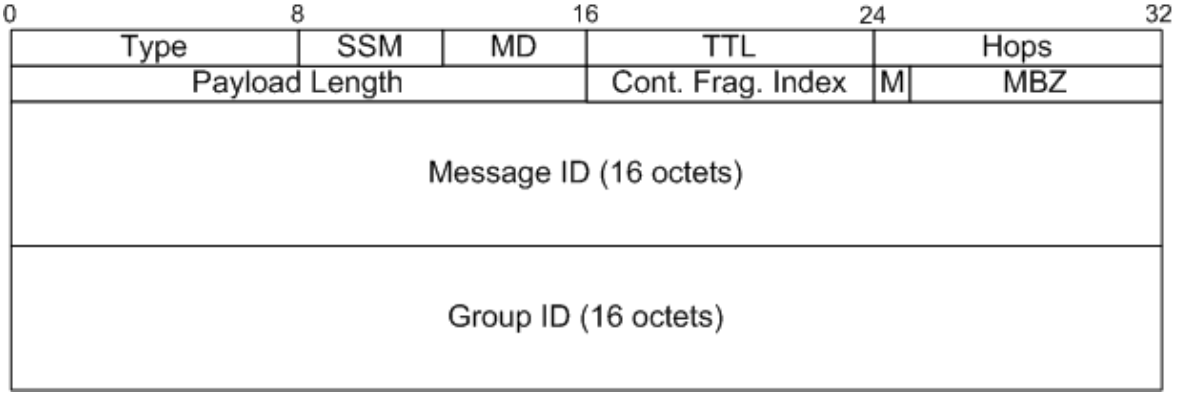

Figure 50. Measurement Group Discovery Request.

The Measurement Group Discovery Request message only comprises the same fields as the common header to all messages. As all the other messages, the message's Type header field is used to identify the message.

A.2.11 Measurement Group Discovery Response

The Measurement Group Discovery Response message comprises the following fields:

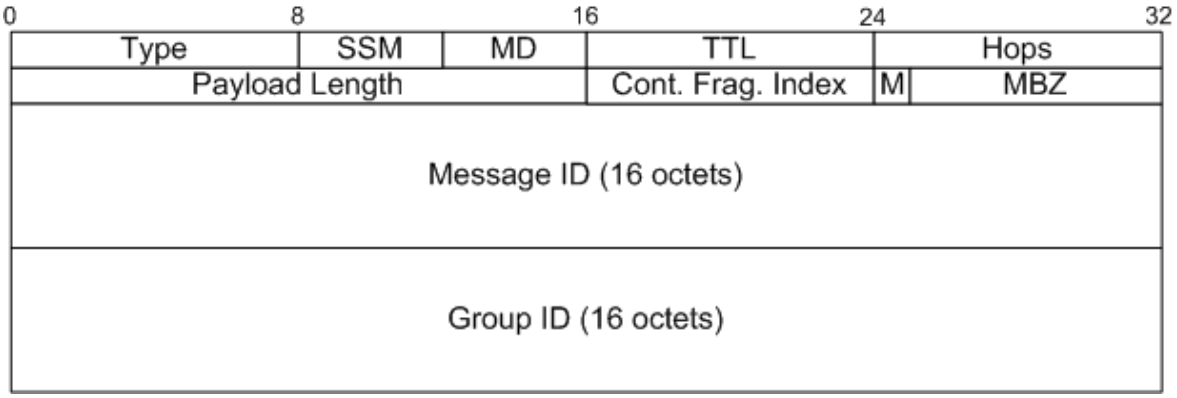

Figure 51. Measurement Group Discovery Response.

The Message ID field of the Measurement Group Discovery Response message must be equal to the Message ID of the Measurement Group Discovery Request message it is sent in reply to.

The Measurement Group Discovery Response message only comprises the same fields as the common header to all messages. In this case, the receiving client will only need to use the information received in the message's header Group ID field (section 4.4.2.1).

A.2.12 List of Nodes Discovery Request

The List of Nodes Discovery Request message comprises the following fields:

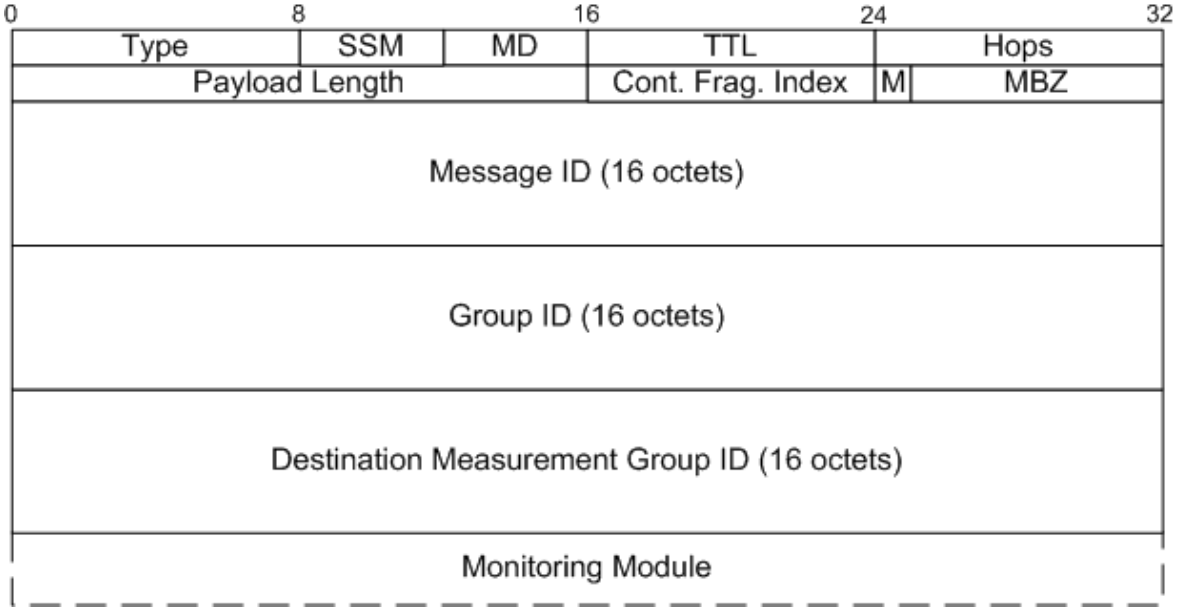

Figure 52. List of Nodes Discovery Request.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group in the network.

The Monitoring Module field represents the name of the monitoring module the user wants to use (section 4.4.2.2.1). Only the nodes that support this monitoring module should be included in the response to this message. This field is transmitted in ASCII format, using the ISO-8859-1 (ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1) character encoding variant. With the message's payload length field it is possible to determine the length of the Monitoring Module field.

This message may not comprise the Monitoring Module field. This situation happens in case the sending client wants to obtain all known nodes of a given measurement group (section 4.4.2.2.2).

In case the sending client wants to obtain the list of all known nodes of the network (from all measurement groups), the message will not comprise the Destination Measurement Group ID and Monitoring Module fields (section 4.4.2.2.3).

A.2.13 List of Nodes Discovery Response

The List of Nodes Discovery Response message comprises the following fields:

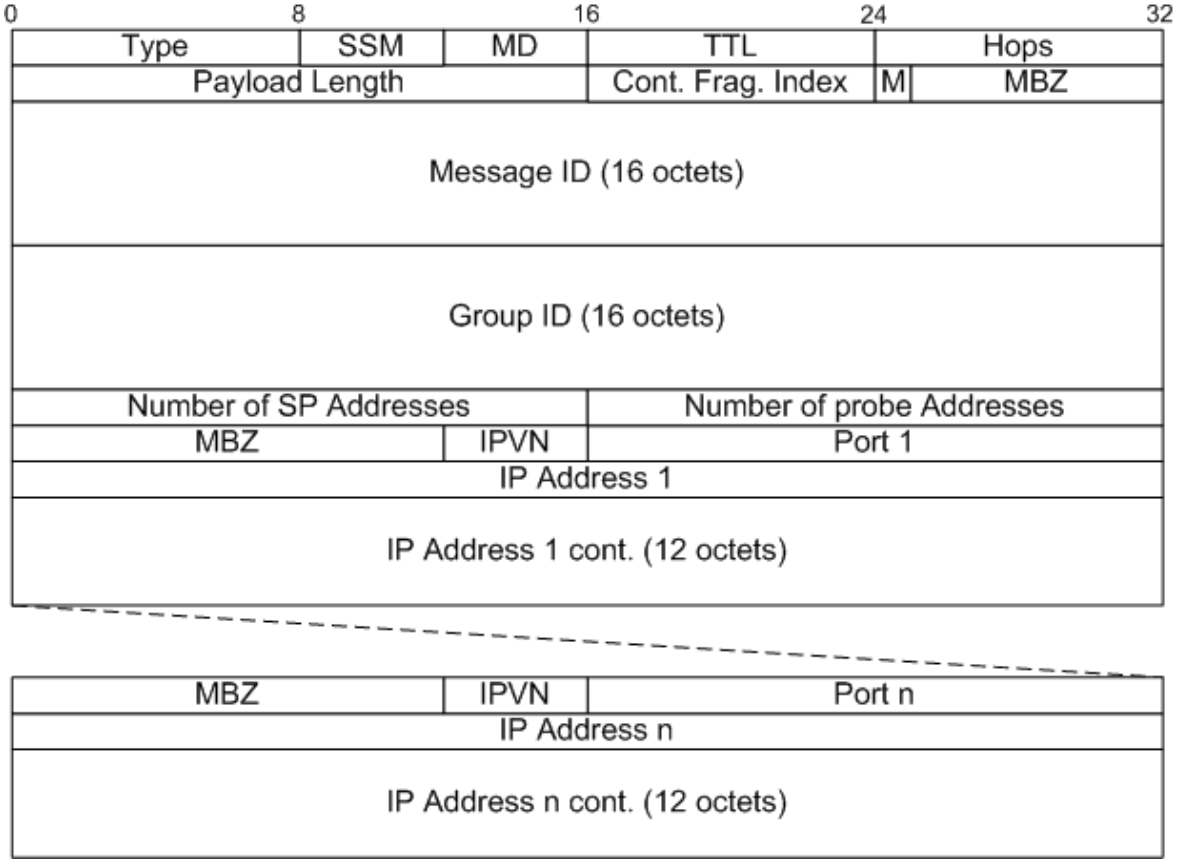

Figure 53. List of Nodes Discovery Response.

The Message ID field of the List of Nodes Discovery Response message must be equal to the Message ID of the List of Nodes Discovery Request message it is sent in reply to.

The List of Nodes Discovery Request message is used by a client to obtain the information about the addresses of the nodes that supports a given monitoring module, or about the addresses of all the nodes connected to a given measurement group, or about the addresses of all the nodes connected to the network (section 4.4.2.2). After receiving a List of Nodes Discovery Request message and whenever required, each receiving super-probe must send to the requesting client the requested information about itself and about the probes connected to it. The List of Nodes Discovery Response message's Number of SP Addresses field is only equal to one when the sending super-probe includes its own address in the message's list of node addresses (should be the first address of the list, if added to it). Otherwise, the Number of SP Addresses field must be set to zero.

The Number of Probe Addresses is the number of probe addresses of the sending super-probe's that will follow in the list of node addresses.

The IPVN, Port and IP Addresses fields comprises the message's list of node addresses. They have the same semantics as in the Ping message.

A.2.14 List of Supported Monitoring Modules Request

The List of Supported Monitoring Modules Request message comprises the following fields:

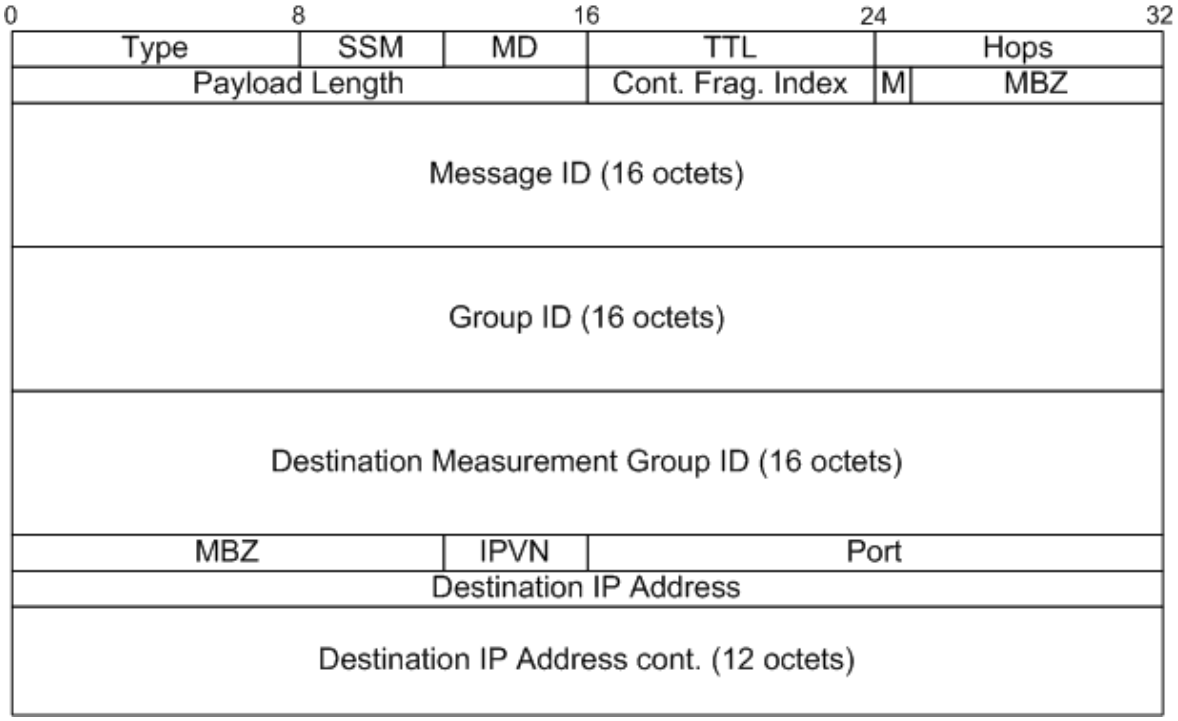

Figure 54. List of Supported Monitoring Modules Request.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Port and Destination IP Address fields has the same semantics of the IPVN, Port and IP Address of the Ping message. They represent the IP address of the node to which this message is destined. It is the node, from which the client wants to get the list of monitoring modules the node is configured to support (section 4.4.2.3).

A.2.15 List of Supported Monitoring Modules Response

The List of Supported Monitoring Modules Response message comprises the following fields:

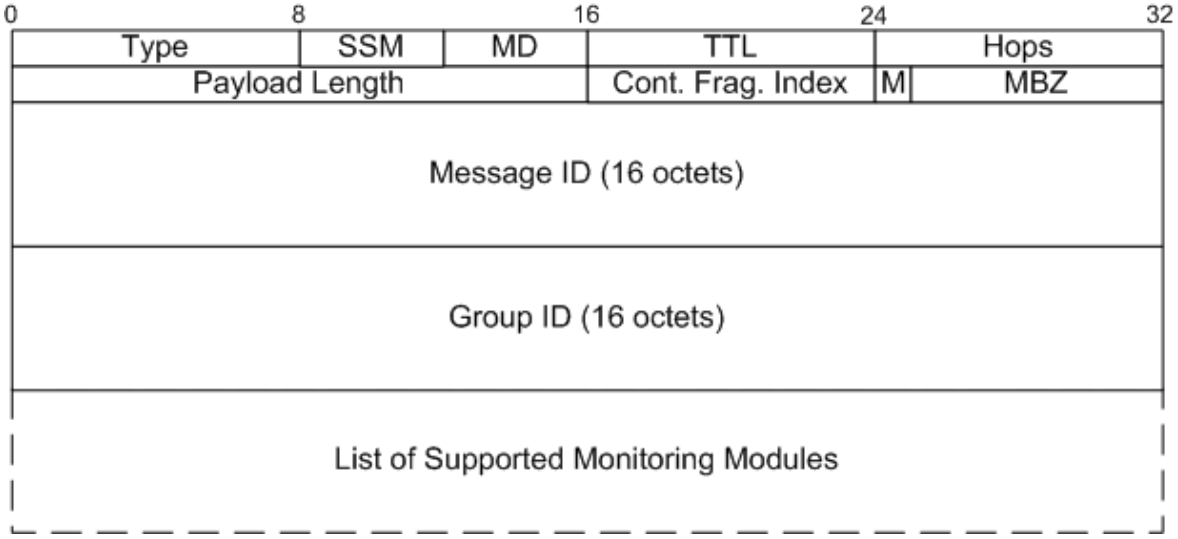

Figure 55. List of Supported Monitoring Modules Response.

The Message ID field of the List of Supported Monitoring Modules Response message must be equal to the Message ID of the List of Supported Monitoring Modules Request message it is sent in reply to.

The List of Supported Monitoring Modules field must comprise the information about the monitoring modules the sending node is configured to support. This field must comprise the content of the File of Supported Monitoring Modules provided by the node's administrator as described in the section 4.4.1.1. For example, it may comprise the following string for a node supporting only the ping monitoring module:

```
<SupportedMonitoringModules>

    <monitoringModule id="ping">
        <name>ping</name>
        <commandToGetHelpDescription>ping</commandToGetHelpDescription>
        <listOfOptionsToSaveToFile></listOfOptionsToSaveToFile>
        <restrictions>
                <mustUse></mustUse>
                <doNotUse>-t</doNotUse>
        </restrictions>
    </monitoringModule>

</SupportedMonitoringModules>
```

This field is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the List of Supported Monitoring Modules field.

A.2.16 Monitoring Module Help Request

The Monitoring Module Help Request message comprises the following fields:

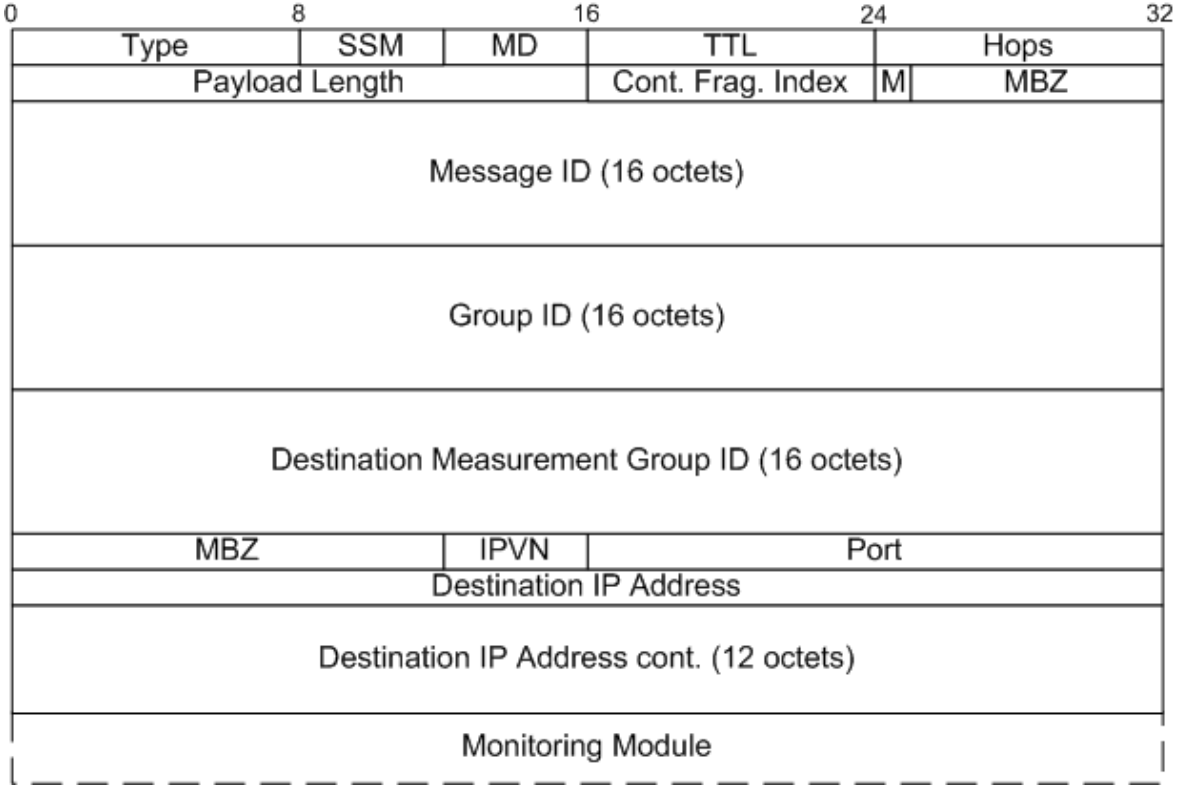

Figure 56. Monitoring Module Help Request.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group in the network.

The IPVN, Port and Destination IP Address fields has the same semantics of the IPVN, Port and IP Address of the Ping message. They represent the IP address of the node to which this message is destined.

The Monitoring Module represents the name of the monitoring module the user wants to get the help or usage description from the destination node (section 4.4.2.4). It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the Monitoring Module field.

A.2.17 Monitoring Module Help Response

The Monitoring Module Help Response message comprises the following fields:

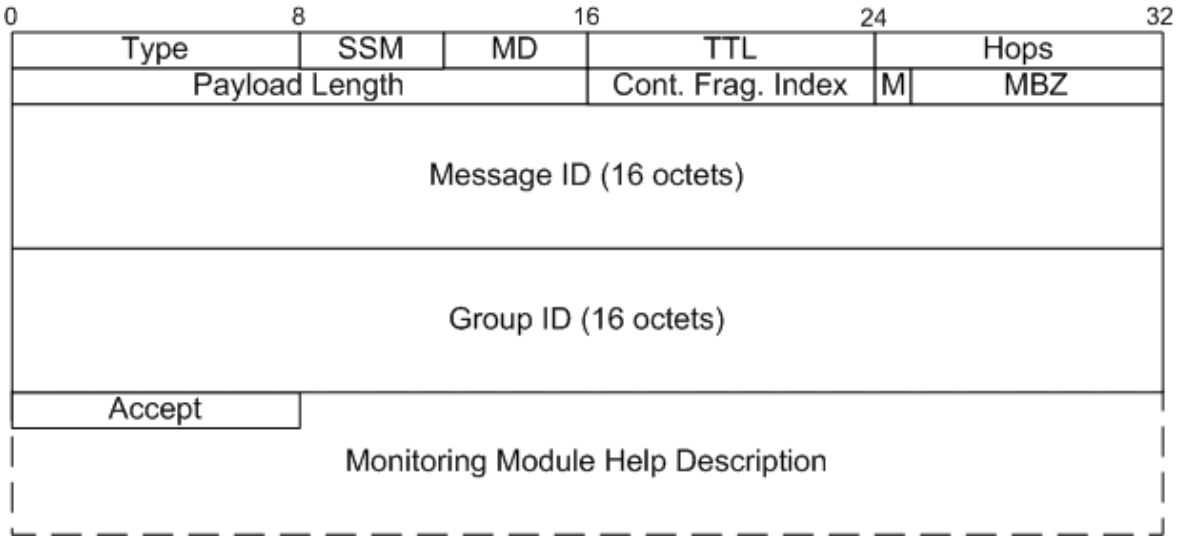

Figure 57. Monitoring Module Help Response.

The Message ID field of the Monitoring Module Help Response message must be equal to the Message ID of the Monitoring Module Help Request message it is sent in reply to.

The message's Accept field is used to identify if the node (probe or super-probe) accepted or not the request. A zero value in the Accept field means that the node accepted the request. Non-zero values indicate that the node does not accept the request.

When the Accept field has a non-zero value it represents the code of the reason the sending node is rejecting the request. The following codes values are meaningful:

1 The node does not support the requested monitoring module;

2 An error occurred while getting the monitoring module's help description.

If the sending node accepts the request, the Monitoring Module Help Description field must comprise the help description or usage for the requested monitoring module (section 4.4.2.4). This field must comprise the information obtained after running the command provided in the File of Supported Monitoring Modules to be used to obtain the help description of the required monitoring module as described in the section 4.4.1.1 (commandToGetHelpDescription tag of the File of Supported Monitoring Modules XML file). This field is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. If the node does not accept the request, the message's Monitoring Module Help Description field must not be sent. With the message's payload length field it is possible to determine the length of the Monitoring Module Help Description field.

A.2.18 Monitoring Module List of Restrictions Request

The Monitoring Module List of Restrictions Request message comprises the following fields:

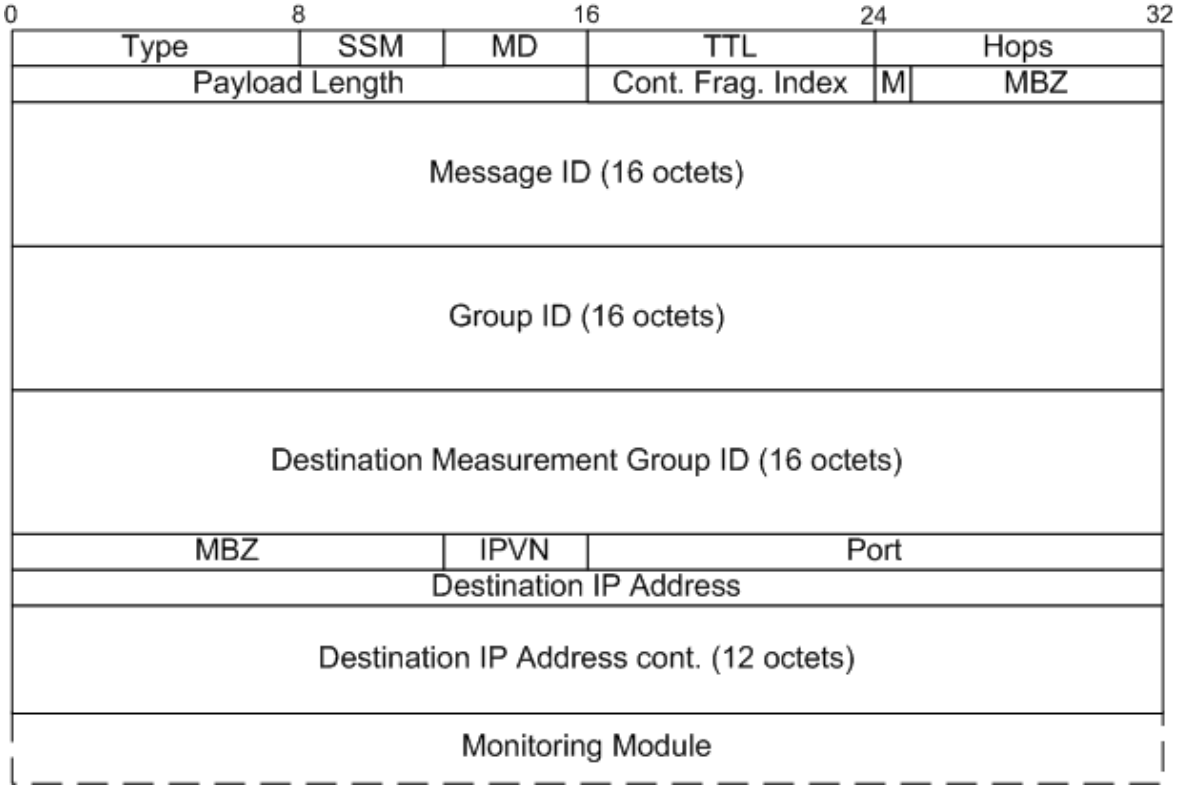

Figure 58. Monitoring Module List of Restrictions Request.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group in the network.

The IPVN, Port and Destination IP Address fields has the same semantics of the IPVN, Port and IP Address of the Ping message. They represent the IP address of the node to which this message is destined.

The Monitoring Module represents the name of the monitoring module the user wants to get the list of restrictions the node is configured for (section 4.4.2.5). It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the Monitoring module field.

A.2.19 Monitoring Module List of Restrictions Response

The Monitoring Module List of Restrictions Response message comprises the following fields:

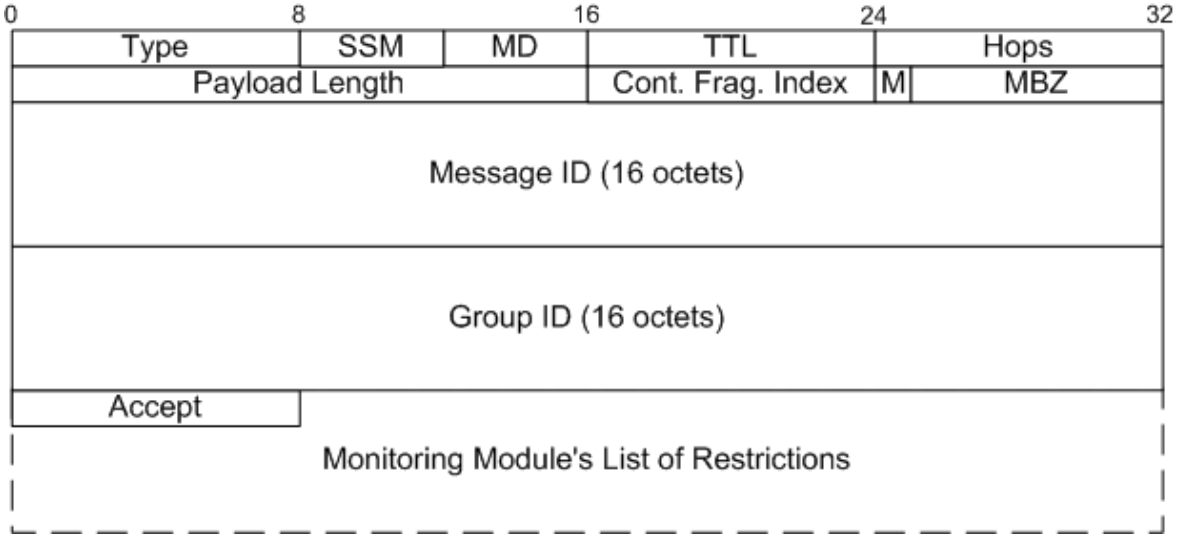

Figure 59. Monitoring Module List of Restrictions Response.

The Message ID field of the Monitoring Module List of Restrictions Response message must be equal to the Message ID of the Monitoring Module List of Restrictions Request message it is sent in reply to.

The message's Accept field is used to identify if the node accepted or not the request. A zero value in the Accept field means that the element accepted the request. Non-zero values indicate that the element does not accept the request.

When the Accept field has a non-zero value it represents the code of the reason the sending element is rejecting the request. The following codes values are meaningful:

1 The node does not support the requested monitoring module;

2 The node does not support a common security mode with the requesting element.

In case the node accepts the request, the Monitoring Module's List of Restrictions field will have the list of restrictions the node is configured for the monitoring module received in the Monitoring Module List of Restrictions Request message. The monitoring module's restrictions are sent as read from the File of Supported Monitoring Modules provided by the node's administrator (section 4.4.1.1). Thus, this field should comprise the string between the XML tag "restrictions" ("*<mustUse></mustUse><doNotUse></doNotUse>*") of the supported monitoring modules XML file. This field is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. If the node does not accept the request, the message's Monitoring Module's List of Restrictions field must not be sent. With the message's payload length field it is possible to determine the length of the Monitoring Module's List of Restrictions field.

A.2.20 Command

The Command message comprises the following fields:

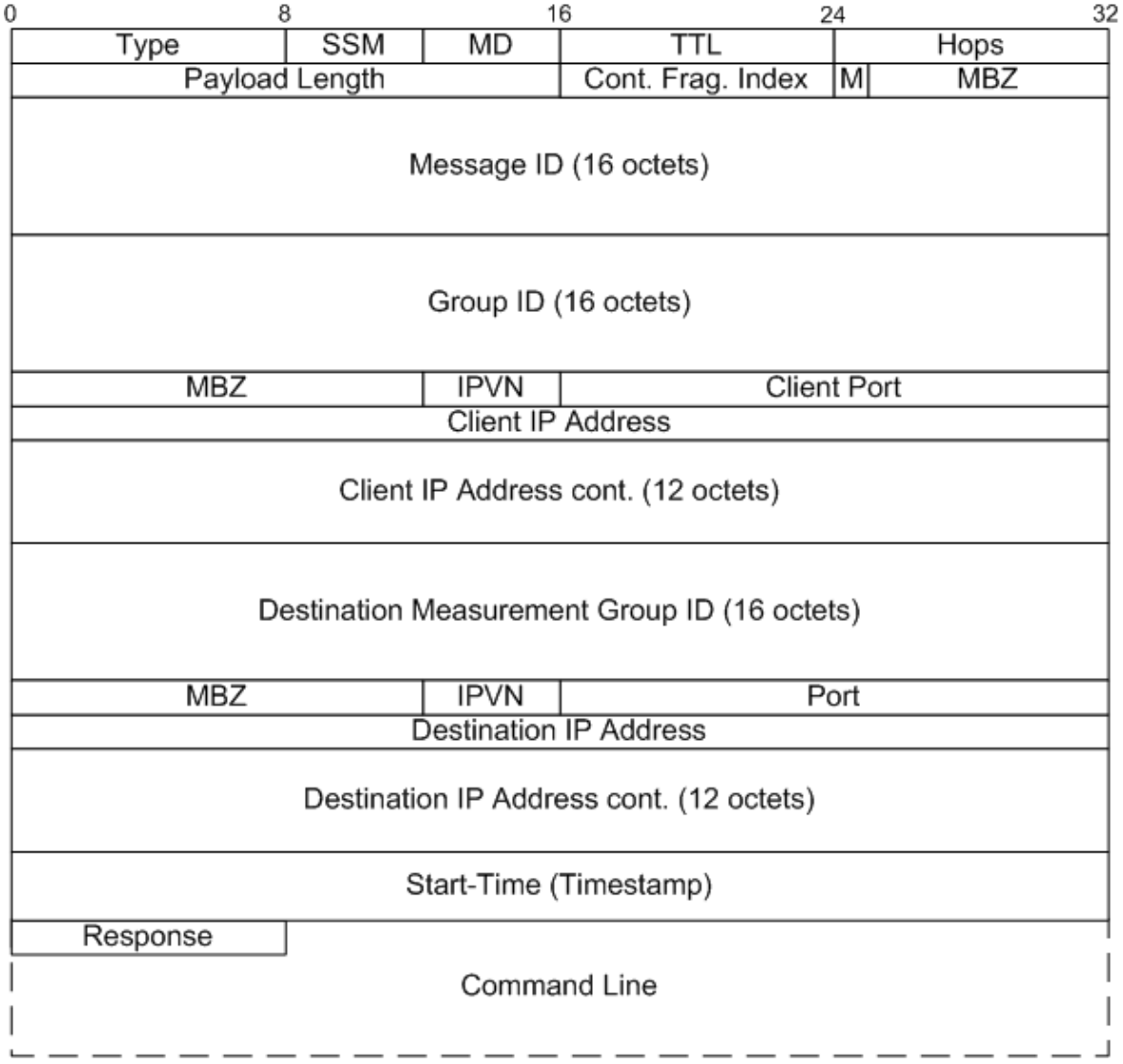

Figure 60. Command.

The IPVN, Client Port and Client IP Address fields has the same semantics of the IPVN, Port and IP Address of the Ping message. They represent the IP address of the client sending this message.

The Destination Measurement Group ID field is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Port and Destination IP Address fields has the same semantics of the IPVN, Port and IP Address of the Ping message. They represent the IP address of the node to which this message is destined.

Start-Time is a timestamp representing the time when the receiving node must start executing the requested command. A Start-Time in the past means that the destination node must immediately start executing the command. The format of the timestamp is the same as in [RFC1305] and is as follows: the first 32 bits represent the unsigned integer number of seconds elapsed since 0h on 1 January 1900; the next 32 bits represent the fractional part of a second that has elapsed since then.

The Response field indicates if the node (super-probe or probe) to which the message is destined should send a response (Command Response message – appendix A.2.21) to the requesting client, after processing the command in this message. With the response message the node can inform the client, if it accepted and successfully processed or not the command received in the Command message. In case the receiving node should send a response to this message, the response field must be set to zero. Any non zero values indicates that no response should be sent.

The Command Line field represents the command line to be used to configure the active/passive measurement test (section 4.4.2.6). It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With this information the node is informed which measurement tool should be used to process the measurement and also the configurations to be used. For example, a possible configuration to a node could be the *ping* command to a given IP address, let's say to the IP *www.ua.pt*, with 10 echo requests. Thus, in the Command Line field must be placed the command line "*ping -n 10 www.ua.pt*" for a node running in a Windows machine. With the message's payload length field it is possible to determine the length of the Command Line field.

A.2.21 Command Response

The Command Response message comprises the following fields:

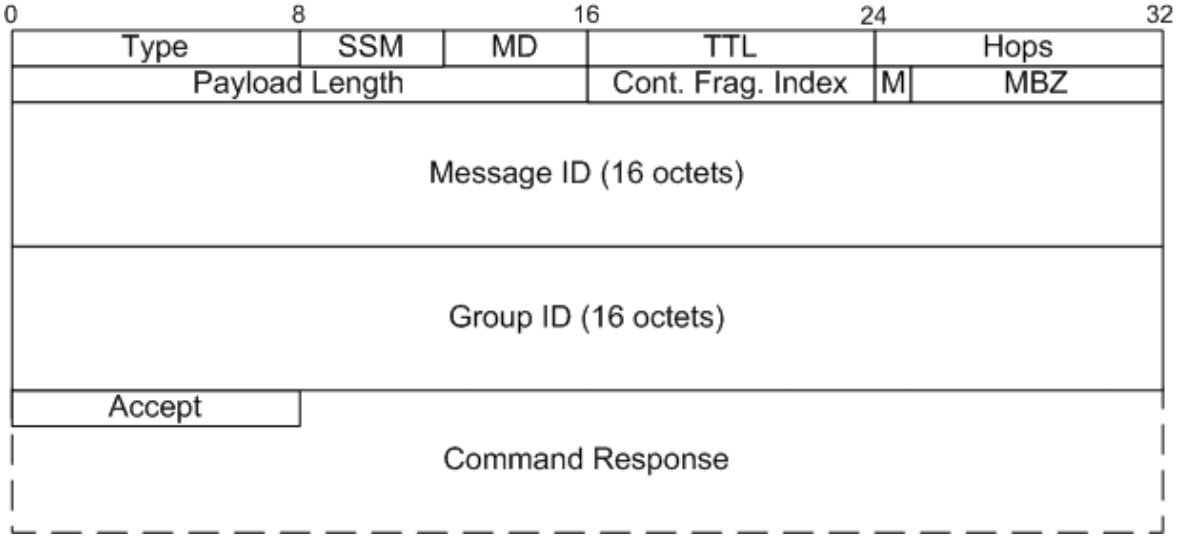

Figure 61. Command Response.

The Message ID field of the Command Response message must be equal to the Message ID of the Command message it is sent in reply to.

The Accept field indicates if the node (super-probe or probe) accepted and successfully processed or not the command received in the Command message. A zero value in the Accept field means that the node accepted and successfully executed the requested command. Any non zero values indicates that the request was rejected.

The Command Response field is the response to the Command message. If the node accepted and successfully processed the command in the received Command message, it has the name of the file where the results of the configured test measurements were saved. If the node rejected the command, it has the reason why the command was rejected. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the Command Response field.

If an error occurs while trying to process the requested command, and the node is supposed to send a Command response message to the requesting client, the node must add to the message the error description of the error occurred. The error description is a string describing why the error occurred. An error may occur in the following situations:

i. The node doesn't support the requested monitoring module or does not support a common security mode with the requesting client;

ii. The received command does not respect the restrictions the node is configured for the requested monitoring module. This error may happen if a required option is not used or the user used an option that should not be used. A user must always follow the restrictions the node is configured for a given monitoring module (section 4.4.1.1);

iii. The node's result directory has reached its maximum size or there is no more free space in the node's storage to store new results file;

iv. Error in the requested command. For example a bad option is in the list of arguments of the requested command. In this case, the output of the execution of the command (the error description) must be sent in the Command Response field of the Command Response message. For example, if a user requests the following command to a node: *ping -d www.ua.pt*. In Windows operating system, the *ping* command doesn't support the option *-d*. In this case, the error output of this command execution, is sent in the Command Response field. For instance, the following error description is sent:

Bad option -d.
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [[-j host-list] | [-k host-list]] [-w timeout] target_name
Options:

-t	*Ping the specified host until stopped. To see statistics and continue - type Control-Break; To stop - type Control-C.*
-a	*Resolve addresses to hostnames.*
-n count	*Number of echo requests to send.*
-l size	*Send buffer size.*
-f	*Set Don't Fragment flag in packet.*
-i TTL	*Time To Live.*
-v TOS	*Type Of Service.*

-r count	*Record route for count hops.*
-s count	*Timestamp for count hops.*
-j host-list	*Loose source route along host-list.*
-k host-list	*Strict source route along host-list.*
-w timeout	*Timeout in milliseconds to wait for each reply.*

v. The monitoring module is not available. This may happen when the node's administrator configured a node to support a given monitoring module, but it is not available.

Sending the description of the error occurred while trying to process a required command is very useful to inform the requesting user why the command was not processed. Otherwise, the user would not be able to determine why a requested command was not processed by the remote node.

A.2.22 Potential Storing Nodes Discovery Request

The Potential Storing Nodes Discovery Request message comprises the following fields:

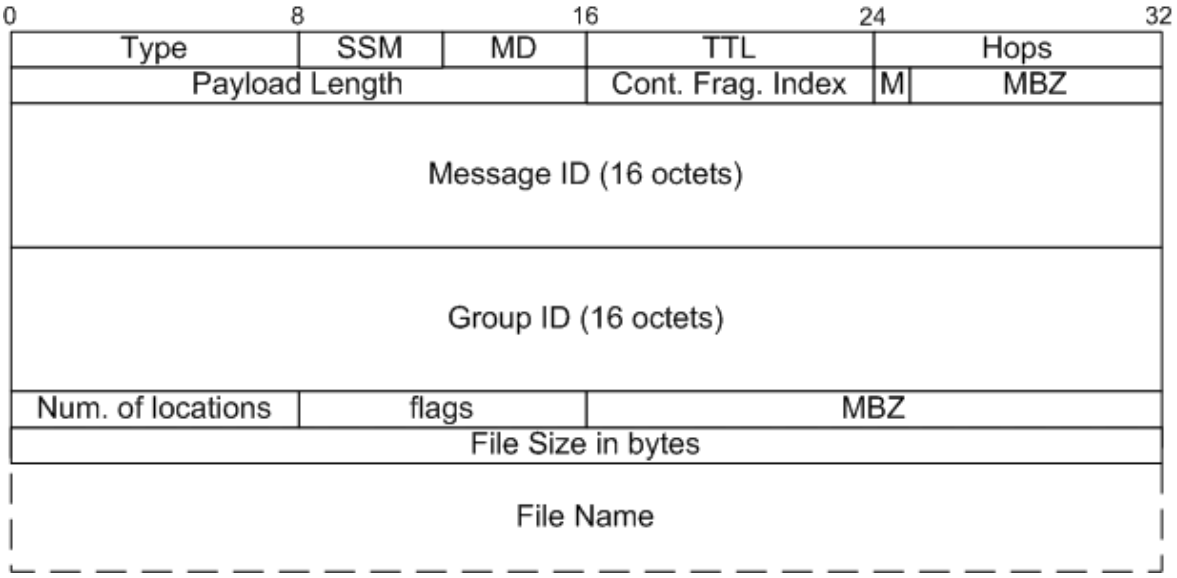

Figure 62. Potential Storing Nodes Discovery Request.

A node may be configured to replicate more than once the Heavy Data Files it generates (section 4.5). The Num. of locations field must have the number of possible locations where the file can be replicated (potential storing nodes) that should be returned.

This message has a one byte field (flags) with the following layout and in the specified order:

bit: Description:

7..3 MBZ (Reserved for future use)

2 flagScope

1 flagPotencialStoringNodesFirewall

0 flagOriginalSourceNodeFirewall

The flagScope field is used to identify the scope of the replication search process. If equal to zero, only nodes of the same measurement group of the (original) source node should answer to the request (local scope). Otherwise, all the nodes receiving the message should answer to the request (global scope).

The flagPotencialStoringNodesFirewall field is used to verify if firewalled locations should or not be returned as potential storing nodes. If equal to zero, firewalled locations can be returned as possible location to replicate the file. When set to 1, it means that only not firewalled locations should be returned. Remember that firewalled nodes can only download from not firewalled locations. Therefore, if the (original) source node is not firewalled or the file has successfully been replicated in a not firewalled location, firewalled locations can be returned as possible location to replicate the file. Otherwise, only not firewalled locations should be returned.

The flagOriginalSourceNodeFirewall field is set (!= 0) if and only if the (original) source node is behind a firewall.

The File Size field is the size of the file to be replicated in bytes.

The File Name field should contain the name of the Heavy Data File to be replicated. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the File Name field.

A.2.23 Potential Storing Nodes Discovery Response

The Potential Storing Nodes Discovery Response message comprises the following fields:

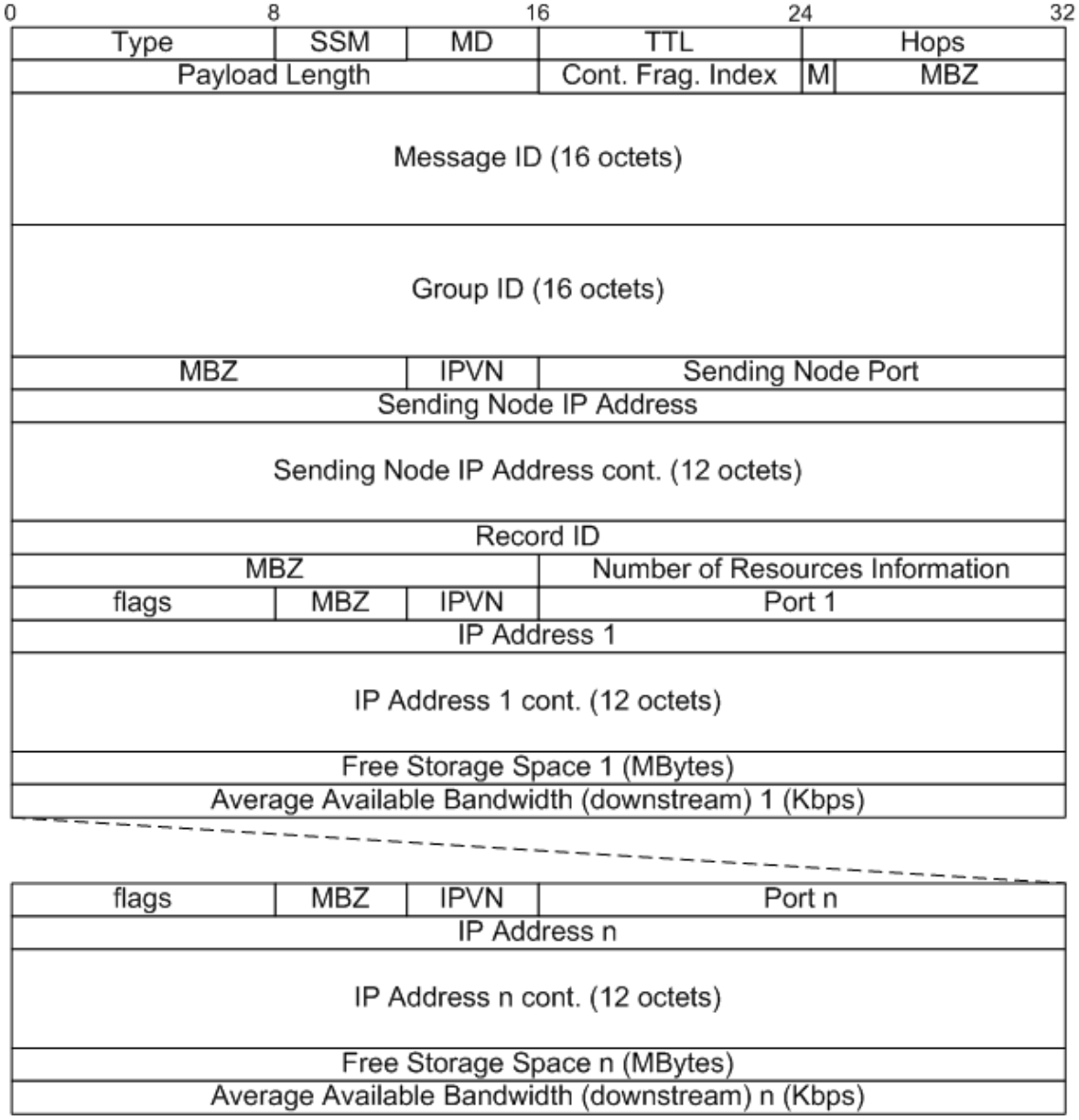

Figure 63. Potential Storing Nodes Discovery Response.

The Message ID of the Potential Storing Nodes Discovery Response message should be equal to the Message ID of the Potential Storing Nodes Discovery Request message it is sent in response to.

The IPVN, Sending Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address and port number where the sending node (probe or super-probe) is running.

The Record ID field must be used by the destination (original) source node to identify to which file replication process the message is related to. It is equal to the hash code (appendix A.3) of the name of the file to be replicated (all lower case). Remember that the Replication Record related to the replication of a given file has a unique ID equal to the file's name hash code (section 4.5.2).

This message may be sent by a probe to inform its super-probe if it is a possible location where a requested file can be replicated, or it is sent by a super-probe to a requesting (original) source node to inform it about the available resources at the probes under its control, that may be possible locations where the requested file can be replicated. The super-probe's available resources may also be included in the Potential Storing Nodes Discovery Response message if it is itself a potential storing node. The Number of Resources Information field is the number of node's Resources Information of potential storing nodes that will follow.

Each node's Resources Information has the following fields:

- flags: is a one byte field with the following layout and in the specified order:

  bit: Description:

  7..2 MBZ (Reserved for future use)

  1 flagDownloadSpeed

  0 flagPush

  The flagDownloadSpeed is set (!= 0) if and only if the Average Available Bandwidth (downstream) field contains the highest average transfer rate (in Kbps) of the last 10 downloads. Otherwise, the Average Available Bandwidth (downstream) field contains the node's total download speed as set by the user, and therefore less reliable.

  The flagPush is set (!= 0) if and only if the node is firewalled or cannot accept incoming TCP connections for any other reason.

- IPVN: has the same meaning as in the Ping message.

- Port: is the port where the corresponding node (probe or super-probe) is waiting for connections.

- IP Address: is the IP address of the corresponding node (probe or super-probe). It must be filled as described in the Ping message.

- Free Storage Space (MBytes): is the node's free storage space that can be used to replicate the file.

- Average Available Downstream Bandwidth (Kbps): is the node's maximum average available downstream network bandwidth.

A.2.24 Replication Request

The Replication Request message comprises the following fields:

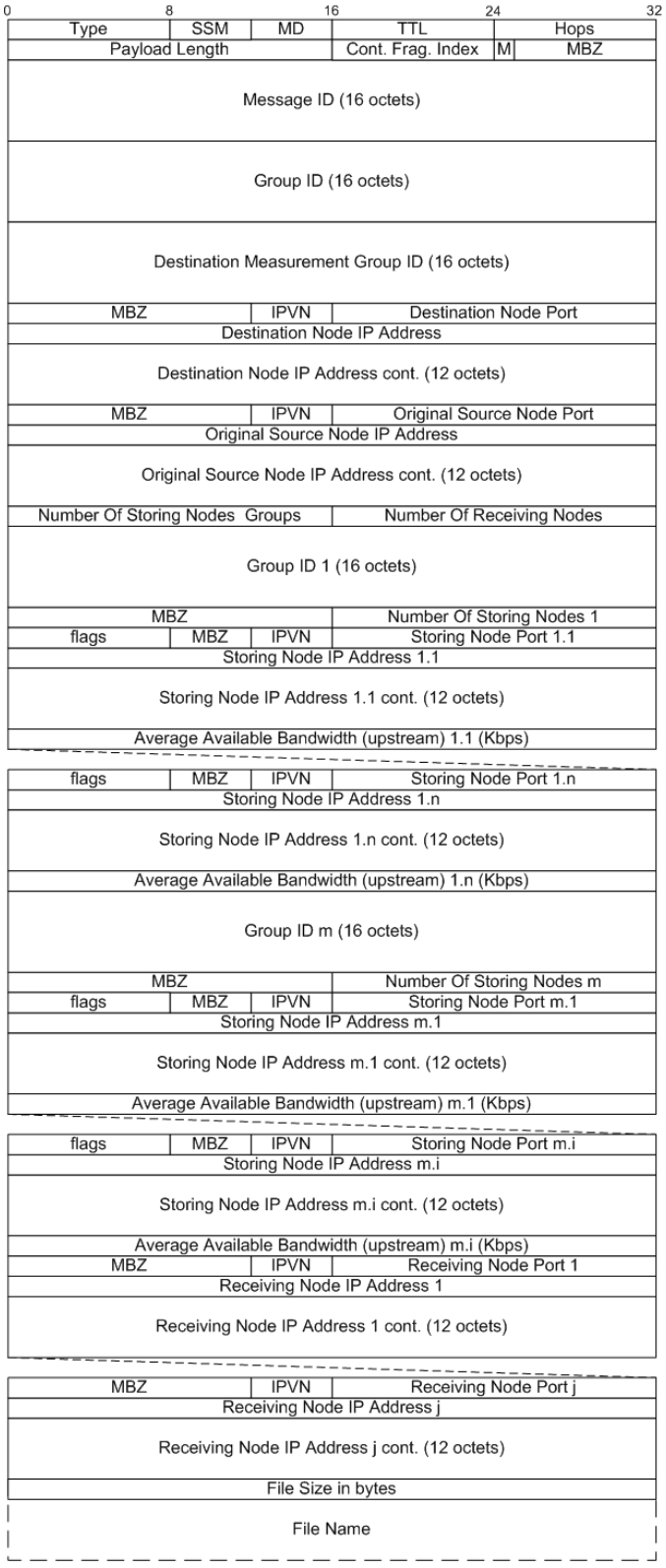

Figure 64. Replication Request.

The Replication Request message must have the information of the node to which it should be forwarded by the super-probes receiving it. The Destination Measurement Group ID represents the Group ID of the measurement group of the node to where the message is destined. It should be used to forward the message to the correspondent measurement group.

The IPVN, Destination Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address and port number of the node to which the message is destined. They should be used to forward the message to the correspondent node.

The IPVN, Original Source Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address and port number of the (original) source node (node requesting the file replication).

The Replication Request message must comprise a list with the information about other nodes from where the file can be retrieved, besides the (original) source node. These nodes are nodes where the file has been successfully replicated before. This is a list of lists, indexed first by measurement group and second by node. This list must contain the information stored in the storingNodesTable of the Replication Record related to the replication of the required file (section 4.5.2). The Number Of Storing Nodes Groups message's field has the number of lists indexed by measurement groups that will follow. Each one of these lists have the information of nodes of a given measurement group. Note: The address of the (original) source node must be in the list related to its measurement group. These lists will have the following fields:

- A Group ID field which represents the Group ID of the measurement group of the nodes which information will follow;
- A Number Of Storing Nodes field which represents the number of nodes storing the file to be replicate at the given measurement group and which information will follow;
- A list of the nodes field with the information of nodes of the given measurement group that are storing the file. This list comprises the following fields:
  - flags: is a one byte field with the following layout and in the specified order:

    bit: Description:

    7..2 MBZ (Reserved for future use)

    1 flagUploadSpeed

    0 flagPush

    The flagUploadSpeed is set (!= 0) if and only if the Average Available Bandwidth (upstream) field contains the highest average transfer rate (in

Kbps) of the node's last 10 uploads. Otherwise, the Average Available Bandwidth (upstream) field contains the node's total upload speed as set by the user, and therefore less reliable.

The flagPush is set (!= 0) if and only if the node is firewalled or cannot accept incoming TCP connections for any other reason.

- IPVN: has the same meaning as in the Ping message.
- Storing Node Port: is the port where the corresponding node (probe or super-probe) is waiting for connections.
- Storing Node IP Address: is the IP address of the corresponding node (probe or super-probe). Must be filled as described in the Ping message.
- Average Available Upstream Bandwidth (Kbps): is the node's maximum average available upstream network bandwidth.

The message must also comprise a list with the IP address of the nodes where the file should be replicated. The Number Of Receiving Nodes field represents the number of IP addresses of nodes that are included in this list. This list must be placed right after the list of lists with the information about the nodes storing the file. The IPVN, Receiving Node Port and IP Address fields of the list of receiving nodes have the same semantics of the IPVN, Port and IP Address fields of the Ping message.

The File Size field has the size in bytes of the file to be replicated.

The File Name field has the name of the Heavy Data File to be replicated. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the File Name field.

A.2.25 Replication-Ack

The Replication-Ack message comprises the following fields:

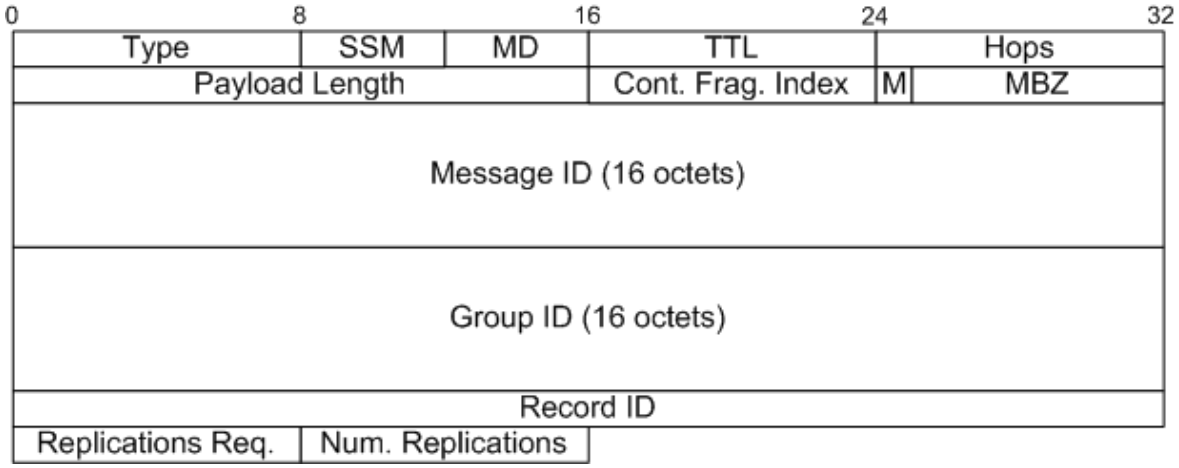

Figure 65. Replication-Ack.

The Message ID of this message should be equal to the Message ID of the Replication Request message sent to initialize the file replication.

The Record ID field must be used by the destination (original) source node to identify to which file replication process the message is related to. It is equal to the hash code (appendix A.3) of the name of the file to be replicated (all lower case). Remember that the Replication Record related to the replication of a given file has a unique ID equal to the file's name hash code (section 4.5.2).

The Replications Req. field represents the number of replications that was requested to the sending node. If the node sending the message is a super-probe, the Replications Req. field has the number of locations where the file should be replicated received in the Replication Request message. If the node sending the message is a probe, the Replications Req. field must be equal to one.

The Num. Replications field stores the number of replications requests that have been accepted.

A.2.26 Download Replication-Ack

The Download Replication-Ack message comprises the following fields:

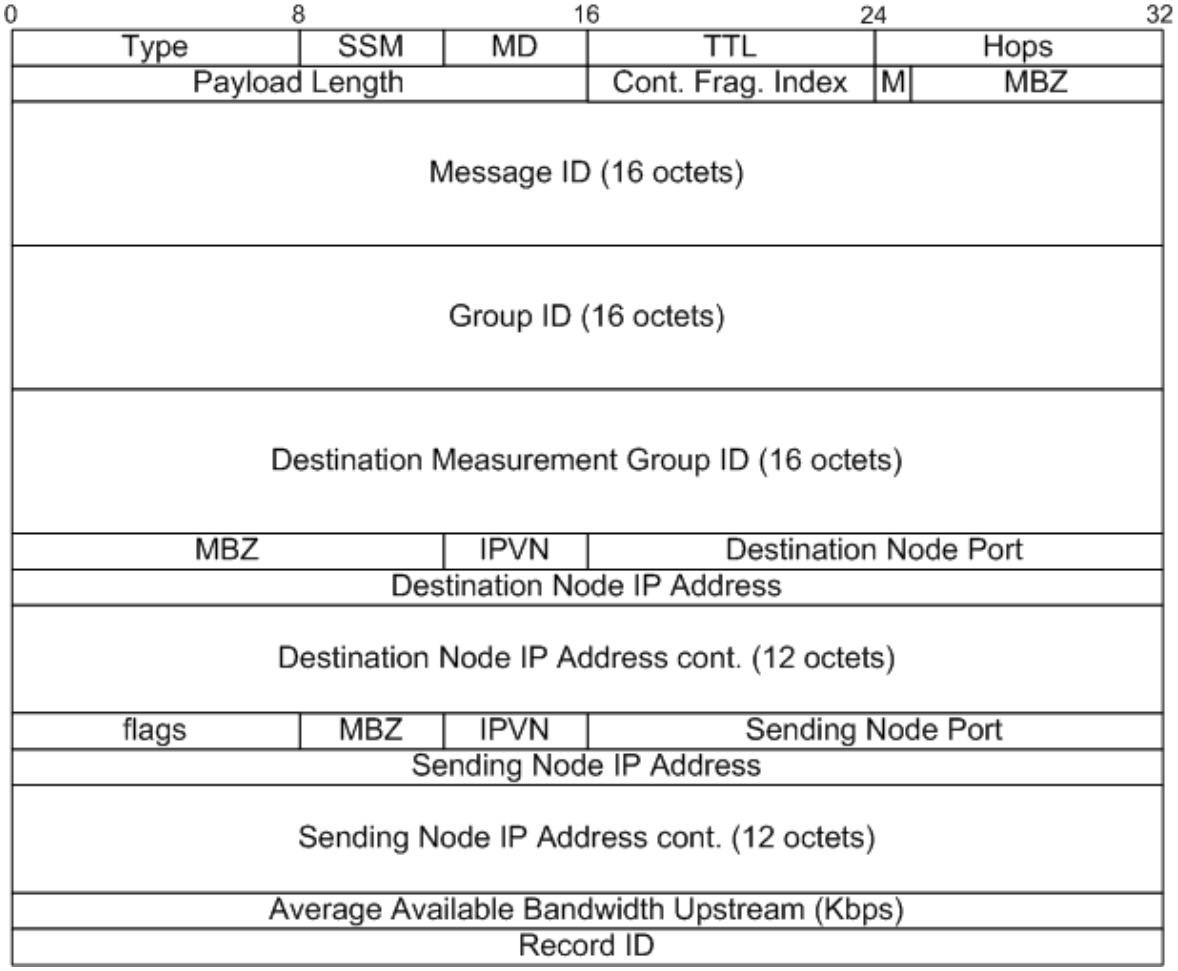

Figure 66. Download Replication-Ack.

The Destination Measurement Group ID field is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network. It must be equal to the Group ID of the (original) source node (node that requested the file replication – section 4.5.3.2.2).

The IPVN, Destination Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which this message is destined. It must be equal to the IP address of the (original) source node.

This message has a one byte field (flags) with the following layout and in the specified order:

bit:	Description:
7..3	MBZ (Reserved for future use)

2 flagAck

1 flagUploadSpeed

0 flagPush

The flagAck field is set (!= 0) if and only if the sending node successfully downloaded the file to be replicated.

The flagUploadSpeed is set (!= 0) if and only if the Average Available Bandwidth (upstream) field contains the highest average transfer rate (in Kbps) of the last 10 uploads. Otherwise, the Average Available Bandwidth (upstream) field contains the node's total upload speed as set by the user, and therefore less reliable.

The flagPush field is set (!= 0) if and only if the sending node is firewalled or cannot accept incoming TCP connections for any other reason.

The IPVN, Sending Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node sending the message.

The Average Available Bandwidth Upstream (Kbps) is the maximum average available network bandwidth (upstream) at the machine where the correspondent element is running.

The Record ID field must be used by the destination (original) source node to identify to which file replication process the message is related to. It is equal to the hash code (appendix A.3) of the name of the file to be replicated (all lower case). Remember that the Replication Record related to the replication of a given file has a unique ID equal to the file's name hash code (section 4.5.2).

A.2.27 Query

The Query message comprises the following fields:

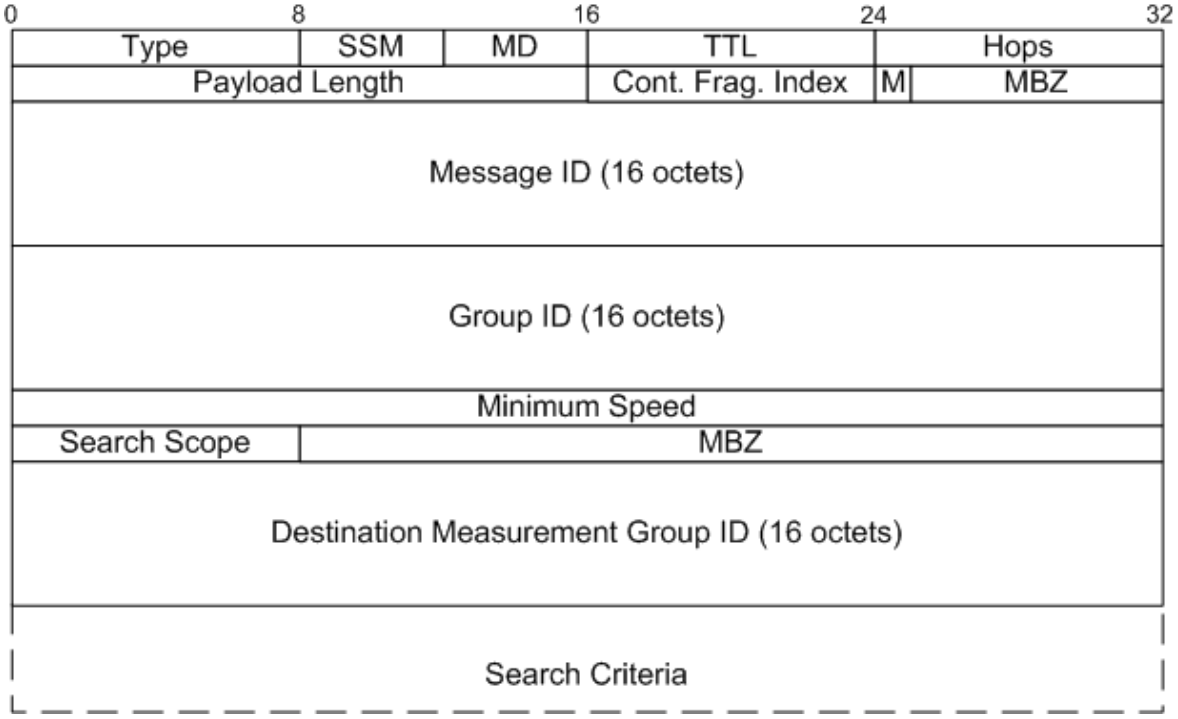

Figure 67. Query.

Minimum Speed is the minimum speed (upload speed in Kbps) of nodes that should be included in the Query-Hit message to be sent in response to this message. A node receiving a Query message, with a Minimum Speed field of n Kbps, should only respond with a Query-Hit message, if it satisfies the Search Criteria and it is able to communicate at a speed >= n Kbps. If this field is set to zero it should be ignored.

Search Scope indicates if a global or a local search should be executed (section 4.6.2). On a global search all super-probes of the system should process the Query message, to determine if they or one of the probes under their control is sharing file(s) that match the search criteria. On a local search, only the super-probes of the measurement group with the same Group ID of the one in the message's Destination Measurement Group ID field, should process the Query message. When the Search Scope is equal to zero a local search is being requested. Any non-zero value in the Search Scope field must be interpreted as a global search request.

The Destination Measurement Group ID field is used to identify the measurement group to which the Query is designated. It is a 16 byte string uniquely identifying the group at the network. It is only meaningful when the Search Scope is equal to 0 (local search).

Search Criteria stores the search criteria. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. The Search Criteria is a string of keywords. A node should only respond with files that has all the keywords. It is recommended to break up the words on any non-alphanumeric characters (anything but letters and numbers). A space is

the standard separator between words. But the complete name of a file may be defined as the Search Criteria. In this last case the "_" and "-"characters are the separator between words. Empty queries or queries containing only 1 letter words should be ignored. Thus, the Search Criteria should contain at least a word, which length is greater than 1 character. With the message's payload length field it is possible to determine the length of the Search Criteria field.

A.2.28 Query-Hit

The Query-Hit message comprises the following fields:

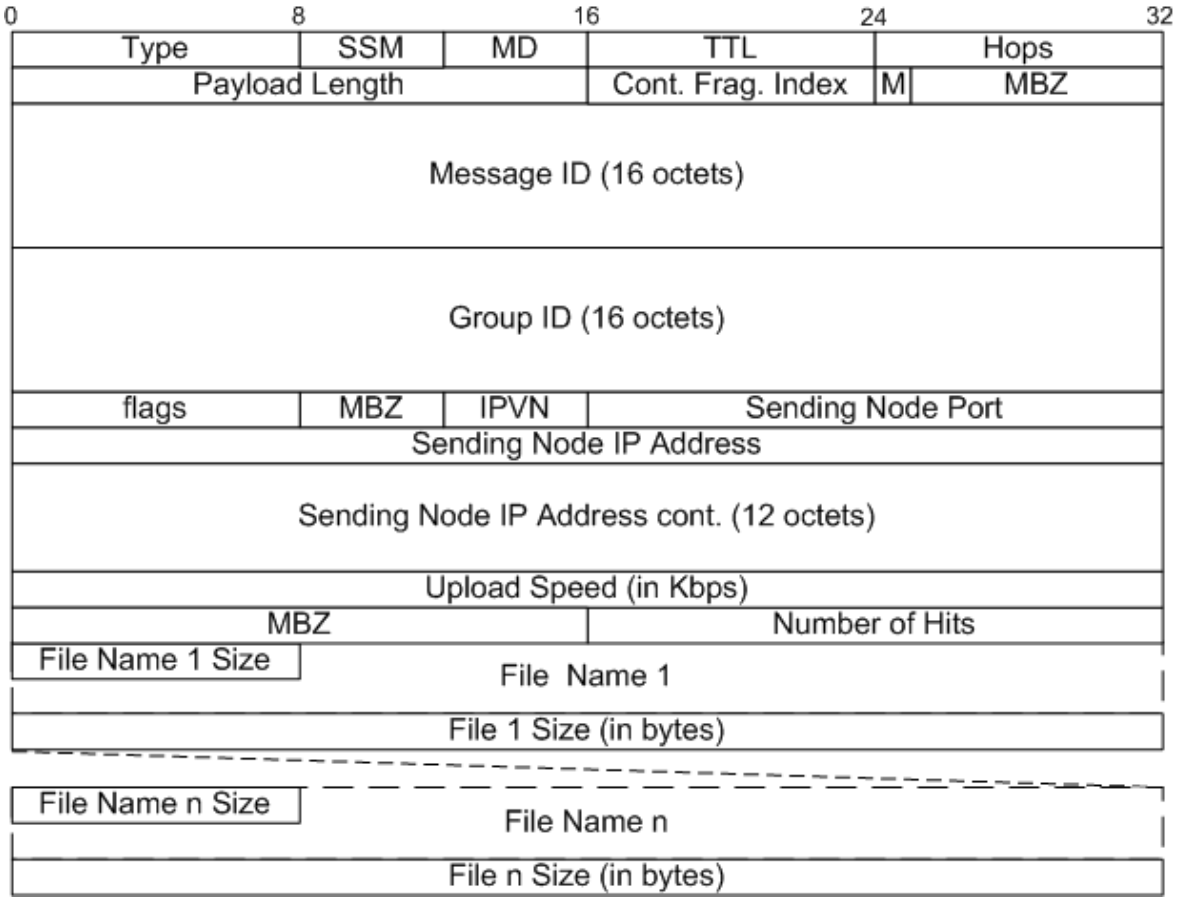

Figure 68. Query-Hit.

The Message ID field of the Query-Hit header should have the same value of the Message ID of the Query message it is send in response to.

This message has a one byte field (flags) with the following layout and in the specified order:

bit:	Description:
7..4	MBZ (Reserved for future use)
3	flagUploadSpeed

2 flagHaveUploaded

1 flagBusy

0 flagPush

The flagUploadSpeed is set (!= 0) if and only if the Upload Speed field of the Query-Hit message contains the highest average transfer rate (in Kbps) of the last 10 uploads. Otherwise Upload Speed field contains the node's total upload speed as set by the user, and therefore less reliable.

The flagHaveUploaded is set (!= 0) if and only if the node has successfully uploaded at least one file.

The flagBusy is set (!= 0) if and only if all of the node's upload/download slots are currently in use (section 4.7).

The flagPush is set (!= 0) if and only if the node is firewalled or cannot accept incoming TCP connection requests.

The IPVN field has the same meaning as in the Ping message.

The Sending Node Port field is the port number on which the responding node can accept incoming HTTP file requests. This is usually the same port as is used for network traffic, but any other port may be used.

The Sending Node IP Address field is the IP Address of the responding node. It must be filled as in the Ping message.

The Upload Speed field is the upload speed (in Kbps) of the responding node.

The Number of Hits field is the number of files the sending node is sharing and that satisfy the search criteria on the received Query message. It represents the number of file hits that will follow in the list of hits field.

Each hit in the list of hits has the following fields:

- File Name Size – this field has the size in bytes of the next field (File Name);
- File Name – this field has the name of a file that satisfies the search criteria of the received Query message. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant;
- File Size – this field represents the size (in bytes) of the respective file.

A.2.29 Push

The Push message comprises the following fields:

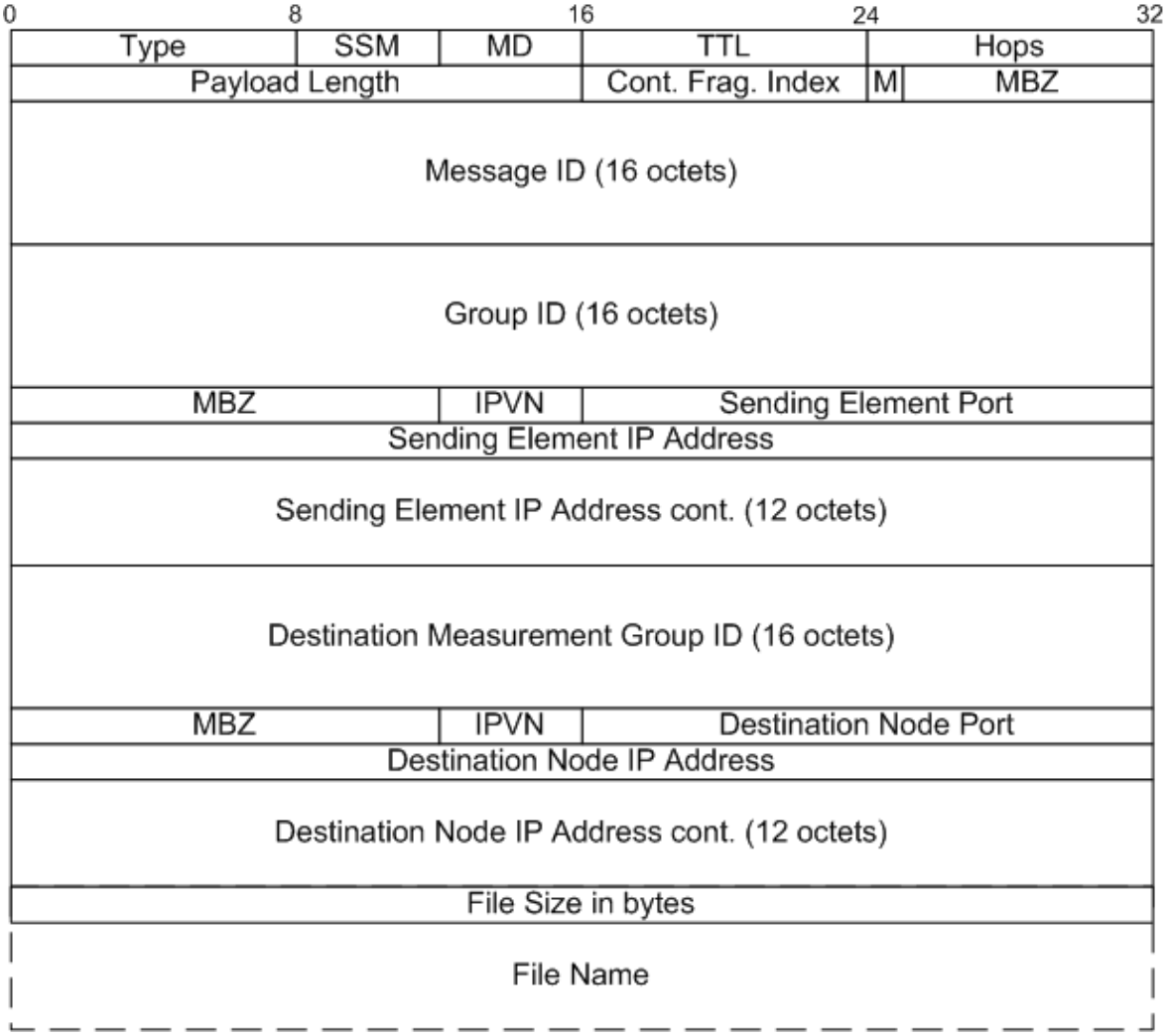

Figure 69. Push.

The IPVN field has the same meaning as in the Ping message.

The Sending Element Port field is the port number on which the sending element can accept incoming connection requests for HTTP file transference. This is usually the same port as is used for network traffic, but any other port may be used.

The Sending Element IP Address field is the IP Address of the sending element. It must be filled as in Ping message.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined (section 4.7.2). It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Destination Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which this message is destined.

The File Size field represents the size (in bytes) of the file the sending element wants to download.

The File Name field has the name of the file the sending element wants to download. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the File Name field.

A.2.30 File Action Request

The File Action Request message comprises the following fields:

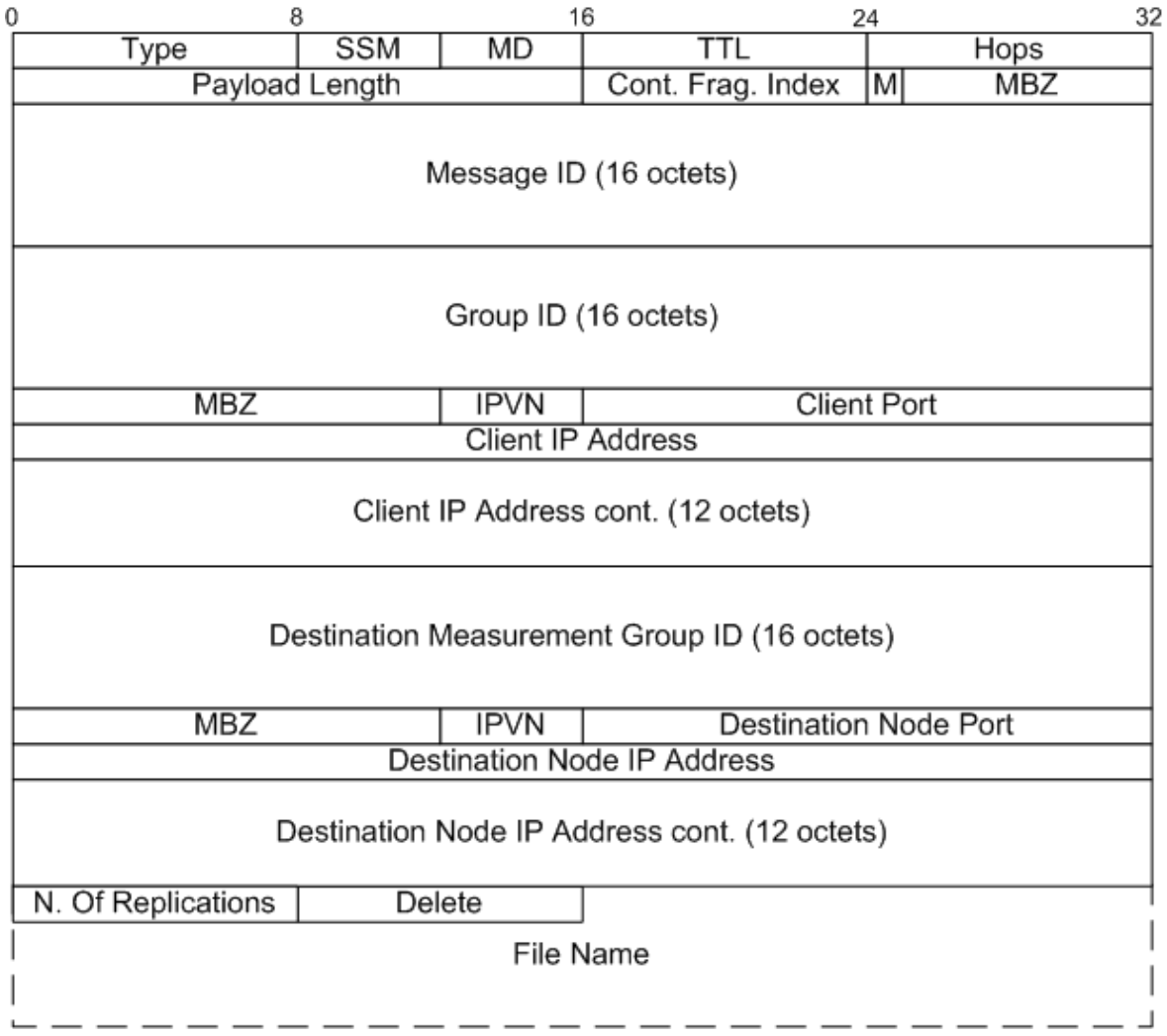

Figure 70. File Action Request.

The IPVN, Client Port and Client IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the client which is sending this message.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Destination Port and Node IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which this message is destined.

The N. Of Replications field is used by the client to request the replication of the file which name is in the File Name field (section 4.9.1). If different than zero it specifies the number of replications that should be performed. Otherwise, the receiving node should not replicate the file.

The Delete field is set (!= 0) if and only if the client is requesting the deletion of the file which name is in the File Name field.

The File Name field has the name of the file to be replicated and/or deleted. It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the File Name field.

A.2.31 File Action Response

The File Action Response message comprises the following fields:

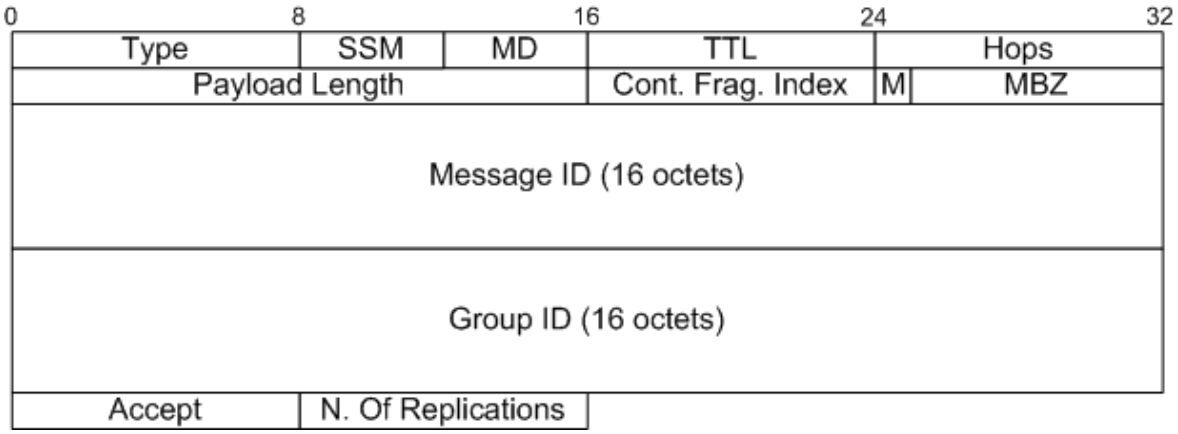

Figure 71. File Action Response.

The Message ID field of the File Action Response message must be equal to the Message ID of the File Action Request message it is sent in reply to.

The Accept field is used to notify the requesting client if the request was or not accepted. If equal to zero it means that the request has been accepted and successfully performed. When the Accept field has a non-zero value it represents the code of the reason the sending element is rejecting the requested action. The following codes values are meaningful:

1 The File Action Request was rejected because a client can only request the deletion or replication of files of test sessions that it configured;

2 The node is not sharing the requested file;

3 The node is already replicating the file;

4 Not all requested replications where successfully performed;

5 Error deleting the requested file.

In case the node was requested to replicate a given file, the N. Of Replications field contains the number of successful replications performed.

A.2.32 Resources Request

The Resources Request message comprises the following fields:

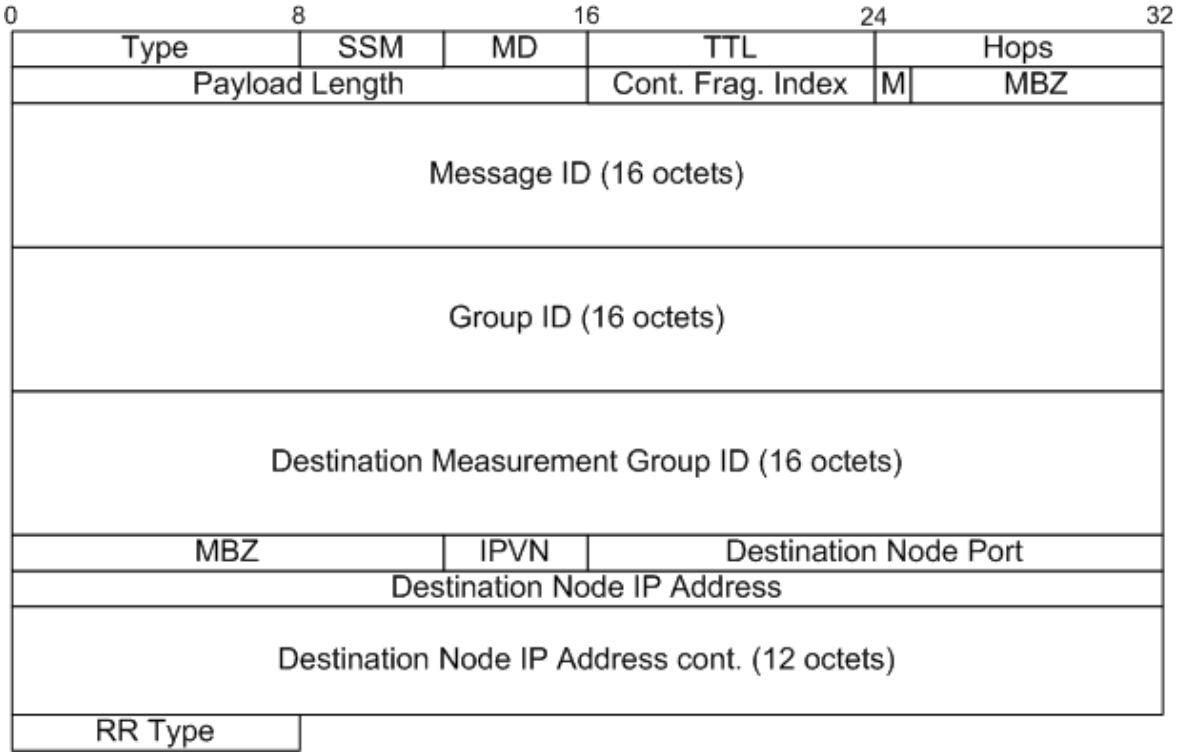

Figure 72. Resources Request.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Destination Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which this message is destined.

The RR Type field is used to identify the type of the Resources Request message (section 4.9.2). The following types are meaningful:

0 When the RR Type field is set to zero, the receiving super-probe should send a Resources message (appendix A.2.33) to the requesting element. It should not forward the received Resources Request message to the probes under its control. In this case, the Resources message, to be sent in response to the received request, should only include the information about the super-probe's available resources;

1 If the RR Type field is equal to one the receiving super-probe should forward this message to all probes under its control. Each probe should reply with a Resources message. Then, with the information received in the Resources messages of each probe, the super-probe should build the Resources messages to be sent to the requesting element. Thus, the Resources message, to be sent in response to the received request, should include the information about the super-probe's available resources and the information of the available resources at the probes under its control.

In the meantime, if the receiving element is a probe, the RR Type field should be ignored. A probe should always send the Resources message case it receives a Resources Request message.

A.2.33 Resources

The Resources message comprises the following fields:

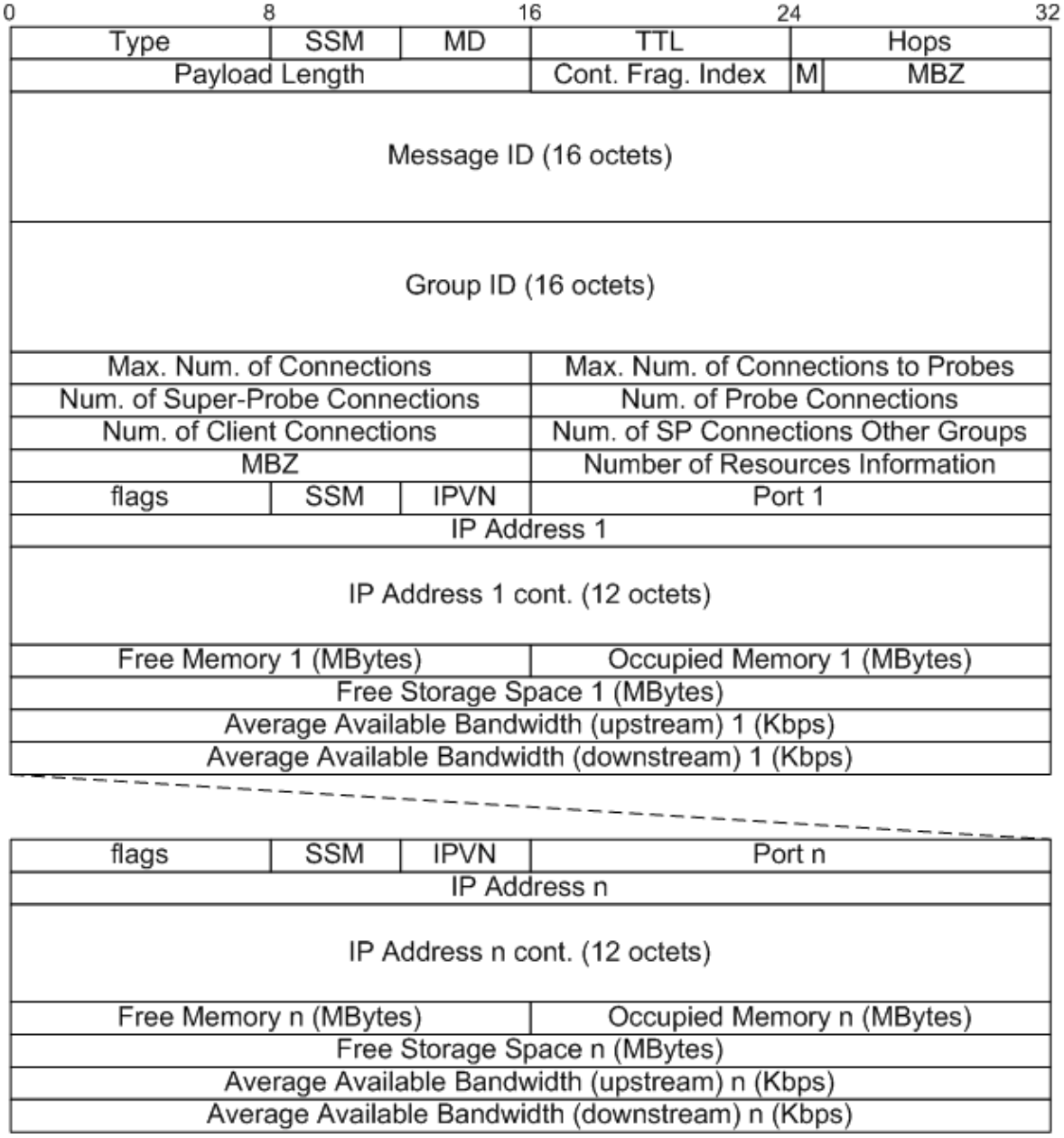

Figure 73. Resources.

The Message ID field of the Resources message must be equal to the Message ID of the Resources Request message it is sent in reply to.

The Max. Num. of Connections field represents the maximum number of connections the sending node can accept, as configured by the node's administrator.

The Max. Num. of Connections to Probes field represents the maximum number of connections to probes of the sending node's measurement group the node can accept, as configured by the node's administrator.

The Num. of Super-Probe Connections field represents the current sending node's number of connections to super-probes of its measurement group.

The Num. of Probe Connections field is the number of connections to probes that the sending node currently has active.

The Num. of Client Connections field is the number of connections to clients that the sending node currently has active.

The Num. of SP Connections Other Groups field represents the current sending node's number of connections to super-probes of other measurement groups.

The Number of Resources Information field is the number of node's Resources Information that will follow. The first Resources Information is always the Resources Information of the sending node. When the sending node is a super-probe and it is supposed to include in the message, the information about the available resources at the probes under its control, the next Resources Information contain information about these probes.

Each node's Resources Information has the following fields:

- flags: is a one byte field with the following layout and in the specified order:

  bit: Description:

  7..3 MBZ (Reserved for future use)

  2 flagUploadSpeed

  1 flagDownloadSpeed

  0 flagPush

  The flagUploadSpeed is set (!= 0) if and only if the Average Available Bandwidth (upstream) field contains the highest average transfer rate (in Kbps) of the last 10 uploads. Otherwise, the Average Available Bandwidth (upstream) field contains the node's total upload speed as set by the node's administrator, and therefore less reliable.

  The flagDownloadSpeed is set (!= 0) if and only if the Average Available Bandwidth (downstream) field contains the highest average transfer rate (in Kbps) of the last 10 downloads. Otherwise, the Average Available Bandwidth (downstream) field contains the node's total download speed as set by the node's administrator, and therefore less reliable.

The flagPush is set (!= 0) if and only if the node is firewalled or cannot accept incoming TCP connections for any other reason.

- The SSM field is used to identify the security modes the sending node supports (supported security modes). The following security mode values are meaningful: 1 for unauthenticated, 2 for authenticated and 4 for encrypted. The value of the SSM field sent is the bit-wise OR of the security mode values that the sending node is willing to support. Thus, the last three bits of the SSM 4-bit value are used. The first bit must be zero. The receiving element must ignore the values in the first bit of the SSM value. This way, this bit is available for future protocol extensions.

- IPVN: has the same meaning as in the Ping message.

- Port: is the port where the corresponding node (probe or super-probe) is waiting for incoming connection requests.

- IP Address: is the IP address of the corresponding node (probe or super-probe). It must be filled as in the Ping message.

- Free Memory (MBytes): is the current node's total available free memory measured in Mbytes.

- Occupied Memory (MBytes): is the amount of memory the node's process is occupying (measured in Mbytes).

- Free Storage Space (MBytes): is the current value of the free storage space (the sending node can use) at the machine where the correspondent node is running.

- Average Available Bandwidth (Kbps): is the maximum average available network bandwidth (upstream/downstream) at the machine where the correspondent node is running.

A.2.34 Light Data

The Light Data message comprises the following fields:

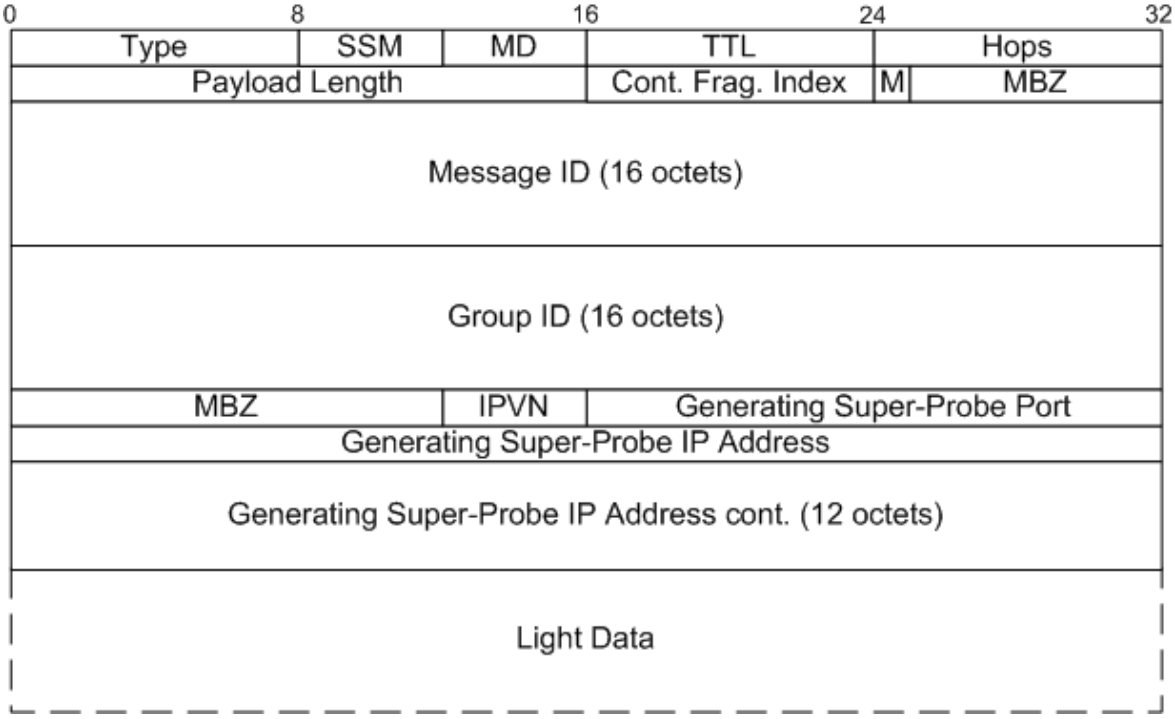

Figure 74. Light Data.

The IPVN, Generating Super-Probe Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the super-probe where the Light Data message was generated.

The Light Data field will have the content of the super-probe's Light Data File to be sent (section 4.8). It is transmitted in ASCII format, using the ISO-8859-1 character encoding variant. With the message's payload length field it is possible to determine the length of the Light Data field.

A.2.35 Connect to Node Request

The Connect to Node Request message comprises the following fields:

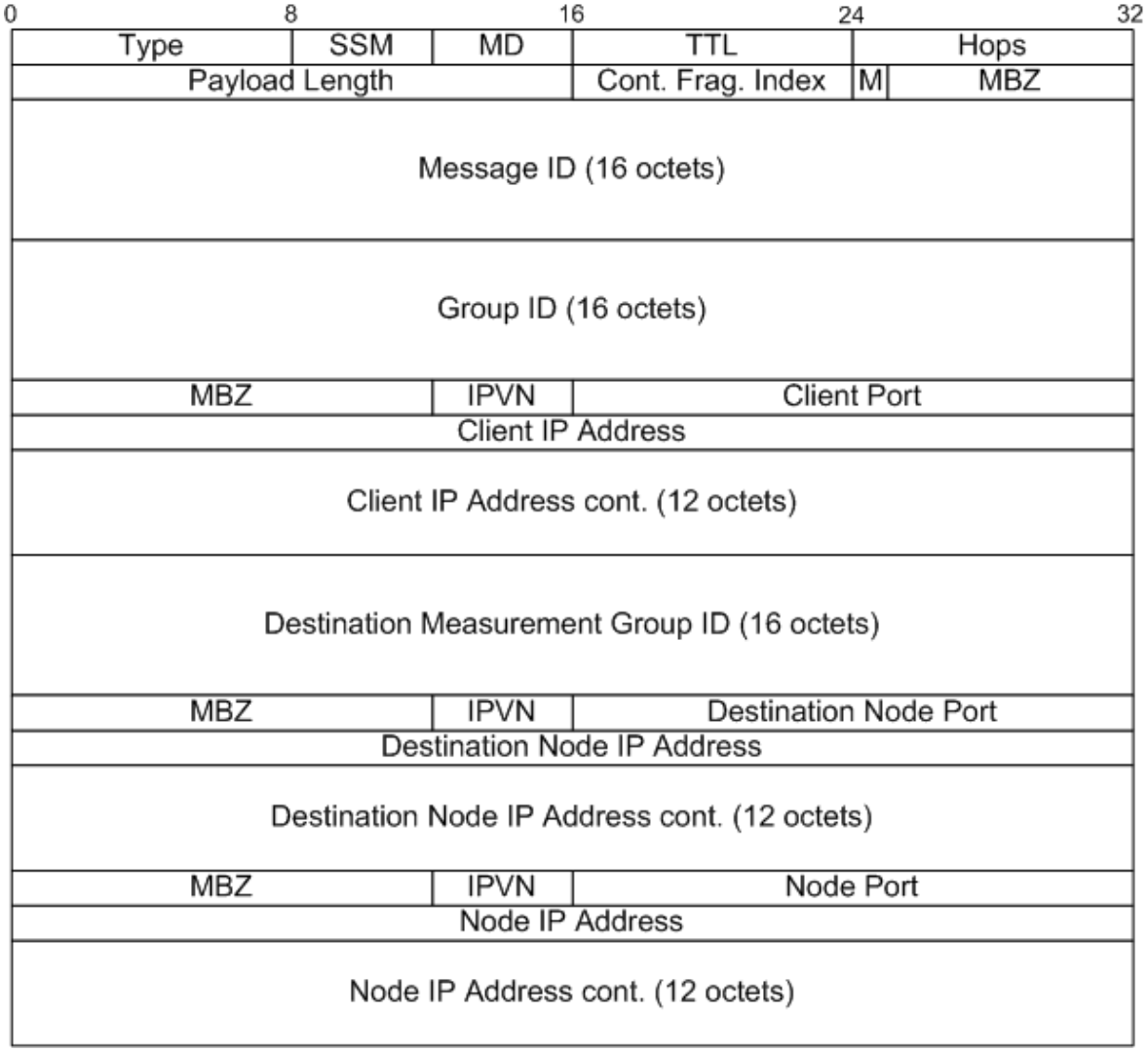

Figure 75. Connect to Node Request.

The IPVN, Client Port and Client IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the client which is sending this message.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network.

The IPVN, Destination Port and Node IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which this message is destined.

The IPVN, Node Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the node to which the destination node should try to connect to (section 4.9.3).

A.2.36 Connect To Node Response

The Connect to Node Response message comprises the following fields:

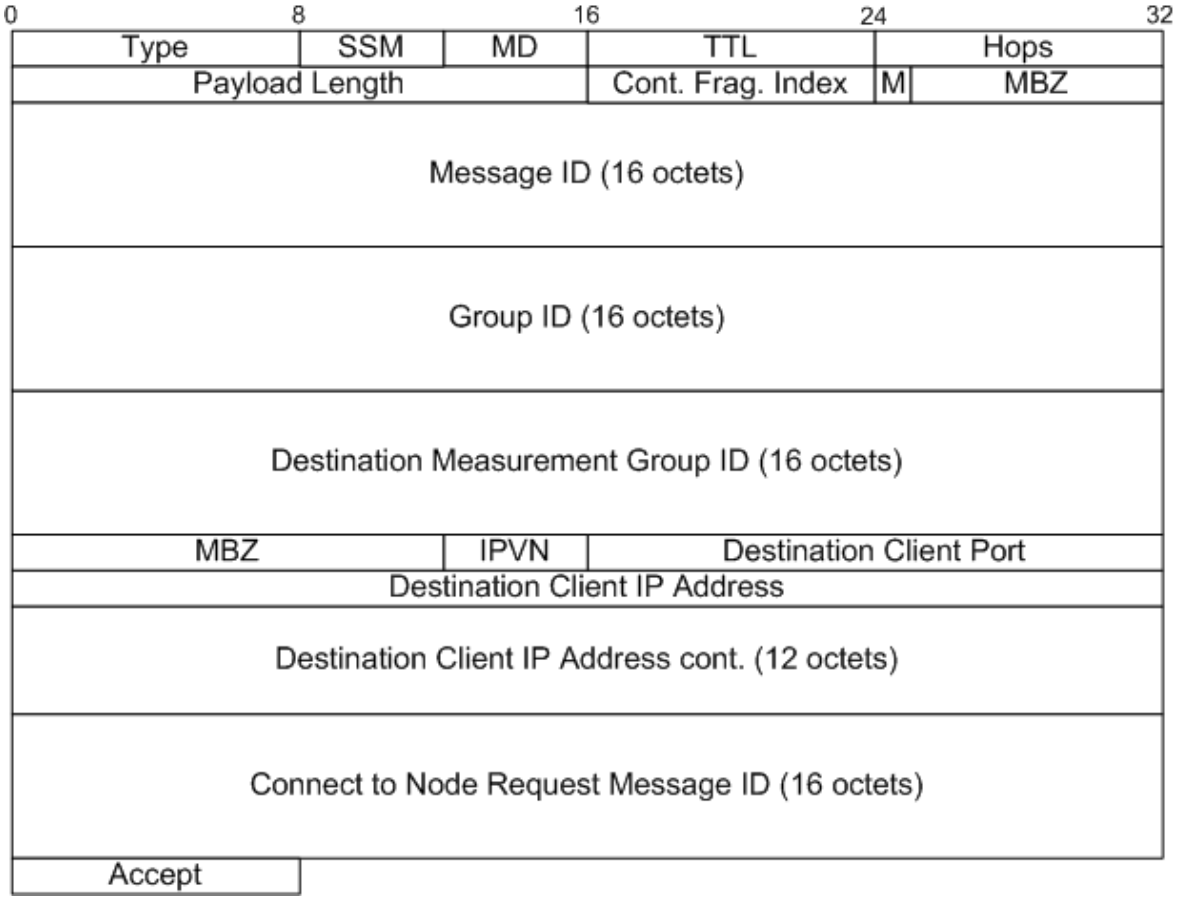

Figure 76. Connect to Node Response.

The Destination Measurement Group ID is used to identify the measurement group to which this message is destined. It is a 16 byte string uniquely identifying the group at the network. It should be equal to the Group ID of the measurement group of the client to which this message is destined.

The IPVN, Destination Client Port and IP Address fields have the same semantics of the IPVN, Port and IP Address fields of the Ping message. They represent the IP address of the client to which this message is destined.

The Connect to Node Response message should be forwarded to the requesting client using the client's IP address and measurement group. This is because it is not always possible to send the message using the same path the Connect to Node Request message was received since the node may be a probe that was requested to connect to another super-probe. Therefore the message should not have the same Message ID of the Connect to Node

Request message it is sent in response to. If it does, it will not be forwarded by the receiving super-probes that previously received the Connect to Node Request message from a different connection (section 4.2.1) because they will detect that they previously received a message with the same Message ID. However this message must still comprise the Message ID of the Connect to Node Request message it is send in response to because the destination client will only process the message if it is the response to the last request it made. The client has this behavior because it may receive a delayed response relative to another Connect to Node Request it made before. This Message ID is sent in the message's Connect To Node Request Message ID field.

The Accept field is used to notify the requesting client if the request was accepted or not. If equal to zero it means that the request has been accepted and successfully performed. When the Accept field has a non-zero value it represents the code of the reason the sending element is rejecting the requested action or the code of the occurred error. The following codes values are meaningful:

1 The node rejected the request because it is already connected to the requested node;

2 The node was not able to connect to the requested node;

3 The node rejected the request because the maximum number of connections, that can be stored in the node's list of connections (LAC – section 4.3.3), has been reached.

A.2.37 Bye

The Bye message is an optional message sent by a probe or super-probe to a node it is directly connected to, to inform of the intention to close the connection and the reason why. A node receiving a Bye message must close the connection immediately. The node that sent the message must wait a few seconds for the remote host to close the connection before closing it. Other data must not be sent after the Bye message.

The Bye message comprises the following fields:

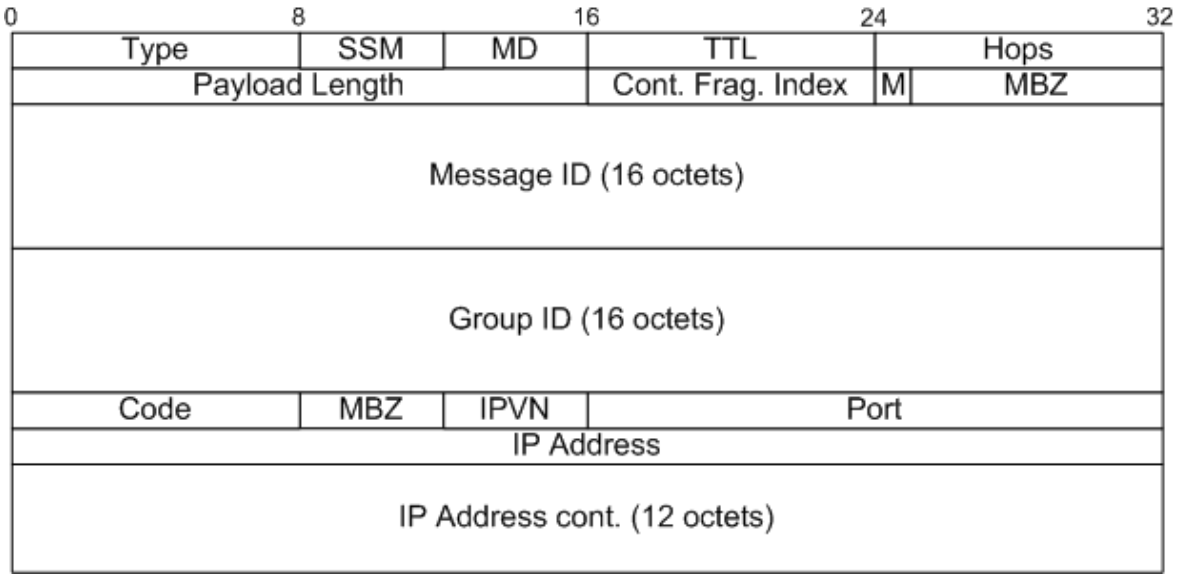

Figure 77. Bye.

The Code field of the Bye message represents the reason the sending element is closing the connection. The following codes are meaningful:

0 The receiving element did nothing wrong, but the sending element chose to close the connection: it is either exiting normally, or the element's administrator requested an explicit close of the connection;

1 This code is used to inform the receiving element that the sending super-probe is demoting itself to a probe (section 4.3.6.2). When this happens, the super-probe must fill the message's IP Address and Port fields with the IP address and port number of the new super-probe to which the receiving element is being assigned;

2 The sending super-probe is overloaded;

3 The receiving element did something wrong, as far as the sending node is concerned: it sent packets deemed too big;

4 The receiving element did something wrong, as far as the sending node is concerned: too many duplicate messages;

5 The receiving element did something wrong, as far as the sending node is concerned: relay improper queries;

6 The receiving element did something wrong, as far as the sending node is concerned: send too many unknown messages;

7 The node noticed an error, but it is an “internal” one: an I/O error or other bad error occurred;

8 The node noticed an error, but it is an “internal” one: a protocol desynchronization;

9 The node noticed an error, but it is an “internal” one: the send queue became full.

The IPVN field has the same meaning of the IPVN field of the Ping message.

The Port field is the port number of the new super-probe to which the receiving element should try to connect to.

IP Address is the IP address of the new super-probe to which the receiving element should try to connect to. It has the same semantics as in the Ping message.

The IPVN, Port and IP address fields are only meaningful if the code 1 is being used. Otherwise, they should not be sent.

A.3 Computing hash codes

In the DTMS-P2P protocol some messages comprise the hash codes of strings to be send over the network. The use of hash codes is advantageous in the sense that smaller messages can be produced because a string hash code only occupies 4 bytes.

In the DTMS-P2P protocol, the hash code for a string object is computed as:

$$s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + ... + s[n-1]$$

using integer arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation.

A.4 XML DOM API and SAX API

There are two possible ways to access XML documents: using DOM API or using the SAX API. The DOM API is based on an entirely different model of document processing than the SAX API. Instead of reading a document one piece at a time (as with SAX), a DOM parser reads an entire document. It then makes the tree for the entire document available to program code for reading and updating. Simply put, the difference between SAX and DOM is the difference between sequential, read-only access, and random, read-write access.

A.5 Test results

Table 18 – DTMS-P2P download speed: 10 KB file on only 1 source (193.136.92.121)

Test	Speed (KB/s)	Delay (sec)
1	268.185	0.037
2	302.245	0.033
3	316.488	0.032
4	319.252	0.031
5	299.526	0.033
6	318.513	0.031
7	277.893	0.036
8	218.712	0.046
9	300.119	0.033
10	292.334	0.034
Average	**291.327**	**0.0346**

Table 19 – DTMS-P2P download speed: 10 KB file only 1 source (193.136.92.228)

Test	Speed (KB/s)	Delay (sec)
1	122.459	0.082
2	286.777	0.035
3	283.134	0.035
4	289.156	0.035
5	279.943	0.036
6	302.125	0.033
7	130.035	0.077
8	253.315	0.039
9	295.366	0.034
10	225.544	0.044
Average	**246.785**	**0.045**

Table 20 – DTMS-P2P download speed: 10 KB file on only 1 source (193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	77.637	0.129
2	82.211	0.122
3	80.442	0.124
4	79.429	0.126
5	62.229	0.161
6	75.990	0.132
7	79.301	0.126
8	73.343	0.136
9	92.139	0.109
10	77.803	0.129
Average	**78.052**	**0.1294**

Table 21 – DTMS-P2P download speed: 10 KB file on only 1 source (193.136.92.234)

Test	Speed (KB/s)	Delay (sec)
1	229.573	0.044
2	198.548	0.050
3	280.529	0.036
4	266.782	0.037
5	269.287	0.037
6	285.475	0.035
7	282.343	0.035
8	248.744	0.040
9	280.059	0.036
10	219.750	0.046
Average	**256.109**	**0.0396**

Table 22 – DTMS-P2P download speed: 500 KB file on only 1 source (193.136.92.121)

Test	Speed (KB/s)	Delay (sec)
1	978.659	0.511
2	978.121	0.511
3	988.719	0.506
4	1013.317	0.493
5	980.610	0.510
6	1009.911	0.495
7	1000.363	0.500
8	992.441	0.504
9	991.449	0.504
10	989.424	0.505
Average	**992.301**	**0.5039**

Table 23 – DTMS-P2P download speed: 500 KB file on only 1 source (193.136.92.228)

Test	Speed (KB/s)	Delay (sec)
1	4690.192	0.107
2	5553.867	0.090
3	4187.484	0.119
4	5976.567	0.084
5	6196.715	0.081
6	6120.219	0.082
7	6285.638	0.080
8	6309.214	0.079
9	6125.058	0.082
10	6285.572	0.080
Average	**5773.053**	**0.0884**

Table 24 – DTMS-P2P download speed: 500 KB file on only 1 source (193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	870.697	0.574
2	821.005	0.609
3	841.783	0.594
4	857.909	0.583
5	863.120	0.579
6	861.680	0.580
7	860.887	0.581
8	863.472	0.579
9	856.039	0.584
10	865.837	0.577
Average	**856.243**	**0.584**

Table 25 – DTMS-P2P download speed: 500 KB file on only 1 source (193.136.92.234)

Test	Speed (KB/s)	Delay (sec)
1	5790.233	0.086
2	5868.588	0.085
3	4375.490	0.114
4	5636.653	0.089
5	4668.720	0.107
6	4405.995	0.113
7	4332.236	0.115
8	4283.341	0.117
9	5526.616	0.090
10	4144.421	0.121
Average	**4903.229**	**0.1037**

Table 26 – DTMS-P2P download speed: 500 KB file on only 2 sources (193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	6281.645	0.080
2	6181.498	0.081
3	6138.566	0.081
4	6229.109	0.080
5	5342.652	0.094
6	6060.944	0.082
7	5629.614	0.089
8	6242.013	0.080
9	6224.928	0.080
10	6267.720	0.080
Average	**6059.869**	**0.0827**

Table 27 – DTMS-P2P download speed: 500 KB file on only 3 sources (193.136.92.121, 193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	4669.792	0.107
2	6200.923	0.081
3	6209.938	0.081
4	6193.049	0.081
5	6224.278	0.080
6	6213.862	0.080
7	6061.375	0.082
8	6247.155	0.080
9	4604.284	0.109
10	5361.570	0.093
Average	**5798.623**	**0.0874**

Table 28 – DTMS-P2P download speed: 500 KB file on 4 sources

Test	Speed (KB/s)	Delay (sec)
1	5673.496	0.088
2	5237.526	0.095
3	6100.132	0.082
4	4471.949	0.112
5	3780.829	0.132
6	5098.965	0.098
7	5291.788	0.094
8	4407.786	0.113
9	4227.333	0.118
10	3848.907	0.130
Average	**4813.871**	**0.1062**

Table 29 – DTMS-P2P download speed: 5120 KB file on only 1 source (193.136.92.121)

Test	Speed (KB/s)	Delay (sec)
1	908.354	5.637
2	908.163	5.638
3	971.532	5.270
4	737.792	6.940
5	973.072	5.262
6	1038.430	4.931
7	1004.477	5.097
8	758.335	6.752
9	1028.979	4.976
10	938.283	5.457
Average	**926.742**	**5.596**

Table 30 – DTMS-P2P download speed: 5120 KB file on only 1 source (193.136.92.228)

Test	Speed (KB/s)	Delay (sec)
1	9255.762	0.553
2	9253.365	0.553
3	8736.041	0.586
4	9231.845	0.555
5	9129.814	0.561
6	8861.112	0.578
7	9292.598	0.551
8	9114.979	0.562
9	9059.151	0.565
10	9229.725	0.555
Average	**9116.439**	**0.5619**

Table 31 – DTMS-P2P download speed: 5120 KB file on only 1 source (193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	995.732	5.142
2	1001.925	5.110
3	938.779	5.454
4	1031.903	4.962
5	987.428	5.185
6	988.277	5.181
7	976.278	5.244
8	1033.147	4.956
9	966.707	5.296
10	807.229	6.343
Average	**972.741**	**5.2873**

Table 32 – DTMS-P2P download speed: 5120 KB file on only 1 source (193.136.92.234)

Test	Speed (KB/s)	Delay (sec)
1	8939.003	0.573
2	9206.630	0.556
3	8853.172	0.578
4	9071.923	0.564
5	8914.957	0.574
6	9051.018	0.566
7	9243.760	0.554
8	9187.850	0.557
9	9075.054	0.564
10	9142.096	0.560
Average	**9068.546**	**0.5646**

Table 33 – DTMS-P2P download speed: 5120 KB file on only 2 sources (193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	7058.115	0.725
2	7593.630	0.674
3	7780.156	0.658
4	7222.288	0.709
5	7586.425	0.675
6	7919.327	0.647
7	7009.728	0.730
8	4549.420	1.125
9	7199.393	0.711
10	6705.394	0.764
Average	**7062.388**	**0.7418**

Table 34 – DTMS-P2P download speed: 5120 KB file on only 2 sources (193.136.92.121 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	1052.989	4.862
2	1021.936	5.010
3	1053.214	4.861
4	1036.389	4.940
5	1034.595	4.949
6	1039.132	4.927
7	1042.833	4.910
8	1033.398	4.955
9	1050.585	4.873
10	1047.278	4.889
Average	**1041.235**	**4.9176**

Table 35 – DTMS-P2P download speed: 5120 KB file on only 3 sources (193.136.92.121, 193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	6771.896	0.756
2	4368.898	1.172
3	6564.721	0.780
4	6650.481	0.770
5	6951.013	0.737
6	6583.246	0.778
7	6469.748	0.791
8	6811.838	0.752
9	4739.963	1.080
10	6665.645	0.768
Average	**6257.745**	**0.8384**

Table 36 – DTMS-P2P download speed: 5120 KB file on 4 sources

Test	Speed (KB/s)	Delay (sec)
1	3318.110	1.543
2	6888.421	0.743
3	3537.893	1.447
4	7138.736	0.717
5	7488.040	0.684
6	3432.130	1.492
7	7042.333	0.727
8	2855.291	1.793
9	3310.444	1.547
10	7560.637	0.677
Average	**5257.204**	**1.137**

Table 37 – DTMS-P2P download speed: 54133 KB file on only 2 sources (193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	5916.315	9.150
2	4213.428	12.848
3	7561.603	7.159
4	7193.978	7.525
5	4208.765	12.862
6	7184.979	7.534
7	8433.032	6.419
8	6218.033	8.706
9	7163.308	7.557
10	6775.720	7.989
Average	**6486.916**	**8.7749**

Table 38 – DTMS-P2P download speed: 54133 KB file on only 2 sources (193.136.92.121 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	1075.800	50.318
2	1076.578	50.282
3	1075.296	50.342
4	1077.054	50.260
5	1077.395	50.244
6	1076.964	50.264
7	1077.784	50.226
8	1077.411	50.243
9	1077.102	50.257
10	1075.745	50.321
Average	**1076.713**	**50.2757**

Table 39 – DTMS-P2P download speed: 54133 KB file on only 3 sources (193.136.92.121, 193.136.92.228 and 193.136.92.219)

Test	Speed (KB/s)	Delay (sec)
1	5614.777	9.641
2	5262.804	10.286
3	7892.106	6.859
4	6658.678	8.130
5	6797.417	7.964
6	7793.247	6.946
7	6536.454	8.282
8	7508.831	7.209
9	7009.694	7.723
10	5612.908	9.644
Average	**6668.692**	**8.2684**

Table 40 – DTMS-P2P download speed: 54133 KB file on 4 sources

Test	Speed (KB/s)	Delay (sec)
1	4960.368	10.913
2	3560.736	15.203
3	6489.959	8.341
4	5114.981	10.583
5	7979.760	6.784
6	7501.987	7.216
7	6267.033	8.638
8	8615.117	6.283
9	6100.478	8.873
10	8443.066	6.411
Average	**6503.349**	**8.9245**

Table 41 – IE and FlashGet download speed: 5120 KB file on only 1 source (193.136.92.121)

Test	Speed (KB/s)	
	IE	FlashGet
1	940	1024
2	853	853.33
3	1024	853.33
4	1024	853.33
5	1024	853.33
6	853	853.33
7	1024	853.33
8	1024	853.33
9	1024	853.33
10	1024	853.33
Average	**981.4**	**870.397**

Table 42 – IE and FlashGet download speed: 5120 KB file on only 1 source (193.136.92.228)

Test	Speed (KB/s)	
	IE	FlashGet
1	5120	1280
2	5120	1280
3	5120	1280
4	5120	1280
5	5120	1280
6	5120	1280
7	5120	1280
8	5120	1280
9	5120	1280
10	5120	1280
Average	**5120**	**1280**

Table 43 – IE and FlashGet download speed: 5120 KB file on only 1 source (193.136.92.219)

Test	Speed (KB/s)	
	IE	FlashGet
1	731	1024
2	1024	853.33
3	1024	1024
4	1024	1024
5	1024	853.33
6	1024	853.33
7	1024	1024
8	1024	1024
9	1024	853.33
10	1024	853.33
Average	**994.7**	**938.665**

Table 44 – IE and FlashGet download speed: 5120 KB file on only 1 source (193.136.92.234)

Test	Speed (KB/s)	
	IE	FlashGet
1	5120	1280
2	5120	1280
3	5120	1280
4	5120	1280
5	5120	1280
6	5120	1280
7	5120	1280
8	5120	1280
9	5120	1280
10	5120	1280
Average	**5120**	**1280**

Table 45 –FlashGet download speed: 5120 KB file on only 2 sources (193.136.92.228 and 193.136.92.219)

Test	FlashGet (KB/s)
1	1706.67
2	1706.67
3	1706.67
4	1706.67
5	1706.67
6	1706.67
7	1706.67
8	1706.67
9	1706.67
10	1706.67
Average	**1706.67**

Table 46 – FlashGet download speed: 5120 KB file on only 2 sources (193.136.92.121 and 193.136.92.219)

Test	FlashGet (KB/s)
1	1024
2	1024
3	1024
4	1024
5	1024
6	1024
7	1024
8	1024
9	1024
10	1024
Average	**1024**

Table 47 – FlashGet download speed: 5120 KB file on only 3 sources (193.136.92.121, 193.136.92.228 and 193.136.92.219)

Test	FlashGet (KB/s)
1	1706.67
2	1706.67
3	1706.67
4	1280
5	1280
6	1706.67
7	1706.67
8	1706.67
9	1706.67
10	1706.67
Average	**1621.336**

Table 48 – FlashGet download speed: 5120 KB file on 4 sources

Test	FlashGet (KB/s)
1	1706.67
2	1706.67
3	1706.67
4	1706.67
5	1706.67
6	1706.67
7	1706.67
8	1706.67
9	2560
10	2560
Average	**1877.336**

Table 49 – FlashGet download speed: 54133 KB file on only 2 sources (193.136.92.121 and 193.136.92.219)

Test	FlashGet (KB/s)
1	1002.45
2	1002.45
3	984.23
4	1002.45
5	1002.45
6	1002.45
7	984.23
8	1021.37
9	1002.45
10	984.23
Average	**998.876**

Table 50 – FlashGet download speed: 54133 KB file on 4 sources

Test	FlashGet (KB/s)
1	2706.62
2	2849.08
3	2004.9
4	2165.3
5	2165.3
6	2849.08
7	2577.73
8	2460.56
9	2706.62
10	2849.08
Average	**2533.427**

Table 51 – LimeWire download speed (54133 KB file)

Test	Download Source Group ID/speed in KB/s			
	b	**b and c**	**c and d**	**b, c and d**
1	2453	2662	2983	3024
2	2704	2562	2632	2356
3	2463	2602	2199	2917
4	2430	2259	2489	2563
5	2084	2502	3024	2752
6	2708	2326	2514	3001
7	1961	2296	2514	2812
8	2115	2156	2547	2669
9	2768	2463	3023	2728
10	2626	2542	2436	1799
Average	**2431.2**	**2437.0**	**2636.1**	**2662.1**

Bibliography

Books

[Eckel2002] B. Eckel, *Thinking in Java*, 3rd ed., Prentice-Hall, 2002. Available at http://www.mindview.net/Books/TIJ/ [April 26, 2007].

[Tanenbaum2007] A. S. Tanenbaum, and M. V. Steen, *Distributed Systems Principles and Paradigms*, 2nd ed., Pearson Prentice Hall, 2007.

[Wilson2002] B. J. Wilson, *JXTA*, 1st ed., New Riders Publishing, 2002. Available at http://www.brendonwilson.com/resources/projects/jxta-book/jxta-pdf.zip [April 26, 2007].

Articles and Documents

[AES] "Advanced Encryption Standard (AES)." Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, November 26, 2001 [April 26, 2007].

[AES_CBC] "Advanced Encryption Standard (AES) in Cipher Block Chaining mode (CBC)." Available at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf, December 2001 [April 26, 2007].

[Andersen2001] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 131-145, Banff, Alberta, Canada, October 2001.

[Balakrishnan2003] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking Up Data in P2P Systems," in *Communications of the ACM*, February 2003, Vol. 46, No. 2.

[BitTorrent2007] "BitTorrent." Available at http://www.bittorrent.org/protocol.html, [April 26, 2007].

[Claffy1999] K. Claffy, T. E. Monk, and D. McRobb, "Internet tomography," in *Nature, Web Matters*, January 1999.

[Claffy2006] K. Claffy, M. Crovella, T. Friedman, C. Shannon, and N. Spring, "Community-Oriented Network Measurement Infrastructure (CONMI) Workshop Report," in *ACM SIGCOMM Computer Communications Review (CCR)*, vol. 36 No 2, Apr 2006, pp. 41-48, Apr 2006.

[Cohen2003] B. Cohen, "Incentives Build Robustness in BitTorrent", in *Workshop on Economics of Peer-to-Peer Systems,* Berkeley, USA, May 2003.

[Corral2003] J. Corral, G. Texier, and L. Toutain, "End-to-end active measurement architecture in ip networks (saturne)," in *Proceedings of Passive and Active Measurement Workshop PAM'03*, 2003.

[Druschel2001] P. Druschel, and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *Proc. of the Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schoss Elmau, Germany, May 2001.

[Finkenzeller2006] M. Finkenzeller, G. Kunzmann, R. Schollmeier, and A. Kirstädter, "Critical-Mass of a Distributed End-System Monitoring Service," in *The 2006 World Congress in Computer Science Computer Engineering, and Applied Computing*, Las Vegas, USA, June 2006.

[Gnutella] "Gnutella protocol." Available at http://rfc-gnutella.sourceforge.net/ [April 26, 2007].

[Gnutella04] "The Annotated Gnutella Protocol Specification v0.4", Available at http://rfc-gnutella.sourceforge.net/developer/stable/index.html [April 26, 2007].

[Gnutella06] T. Klingberg, and R. Manfredi, "Gnutella 0.6." Available at http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, June 2002 [April 26, 2007].

[Grossglauser2001] M. Grossglauser and B. Krishnamurthy, "Looking for science in the art of network measurement," in *Proc. of IWDC Workshop*, 2001.

[LimeWire2002] C. Rohrs, "LimeWire Download Code." Available at http://www.limewire.org/techdocs/downloads.htm, 13 November, 2002 [April 26, 2007].

[Lindroos2003] J. Lindroos, "Peer-to-Peer Content Distribution," M.S. thesis, Dep. of Computer Science, Åbo Akademi University, Turku, Finland, 2003.

[Liu2004] W. Liu, R. Boutaba, and J. W. Hong, "pMeasure: A tool for measuring the internet," in *Proc. of the 2nd Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, San Diego, California, USA, October 2004.

[Liu2005] W. Liu, and R. Boutaba, "pMeasure: A Peer-to-Peer Measurement Infrastructure For the Internet," in *Computer Communications Journal, Special Issue on Monitoring and Measurements of IP Networks*, 2005.

[Menezes1997] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography," in *CRC Press*, 1997.

[NetFlow] White Paper: "Introduction to Cisco IOS NetFlow – A Technical Overview." Available at http://www.cisco.com/en/US/products/ps6601/products_white_paper0900aecd80406232.shtml, February, 2006 [April 26, 2007].

[Pasztor2001] A. Pasztor, and D. Veitch, "High precision active probing for internet measurement," in *Proc. of INET'2001*, 2001.

[Peterson2002] Larry Peterson, Thomas Anderson, David Culler, and Timothy Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proc. of the ACM Workshop on Hot Topics in Networks (HotNets)*, pages 59-64, Princeton, NJ, October 2002.

[Plavec2004] F. Plavec, T. S. Czajkowski. "Distributed Replicated File System Based on FreePastry DHT," Course Project Report, University of Toronto, 2004. Available at: www.eecg.utoronto.ca/~plavec/pub/pastry.pdf [May 15, 2008].

[Rabinovich2006] M. Rabinovich, S. Triukose, Z. Wen, and L. Wang, "DipZoom: The Internet Measurements Marketplace," in *IEEE Global Internet Symposium*, 2006.

[Ratnasamy2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM,* 2001.

[Rocha2007] E. Rocha, H. Veiga, R. Valadas, P. Salvador, and A. Nogueira, "Module for Identifying Internet Applications and its integration in a Peer-to-Peer Measurement Tool," in *Proc. of IADIS International Conference Telecommunications, Networks and Systems, MCCSIS 2007*, Lisbon, Portugal, July 3-5, 2007.

[RFC1305] D. L. Mills, "RFC 1305 – Network Time Protocol (Version 3): Specification, Implementation and Analysis." Available at http://www.ietf.org/rfc/rfc1305.txt, March 1992 [April 26, 2007].

[RFC1757] S. Waldbusser, "RFC 1757 – Remote Network Monitoring Management Information Base." Available at http://www.ietf.org/rfc/rfc1757.txt, February 1995 [April 26, 2007].

[RFC2104] H. Krawczyk, M. Bellare, and R. Canetti, "RFC 2104 – HMAC: Keyed-Hashing for Message Authentication." Available at http://www.ietf.org/rfc/rfc2104.txt, February 1997 [April 26, 2007].

[RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1." Available at http://www.ietf.org/rfc/rfc2616.txt, June 1999 [April 26, 2007].

[RFC2898] B. Kaliski, "RFC 2898 – PKCS #5: Password-Based Cryptography Specification Version 2.0." Available at http://www.ietf.org/rfc/rfc2898.txt, September 2000 [April 26, 2007].

[RFC3917] J. Quittek, T. Zseby, B. Claise, and S. Zander, "RFC 3917 – Requirements for IP Flow Information Export (IPFIX)." Available at http://www.ietf.org/rfc/rfc3917.txt, October 2004 [April 26, 2007].

[RFC4086] D. Eastlake, J. Schiller, and S. Crocker, "RFC 4086 – Randomness Requirements for Security." Available at http://www.rfc-archive.org/getrfc.php?rfc=4086, June 2005 [April 26, 2007].

[RFC4656] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas, "RFC 4656 – A One-way Active Measurement Protocol (OWAMP)." Available at http://www.rfc-editor.org/rfc/rfc4656.txt, September 2006 [April 26, 2007].

[RFC792] J. Postel, "RFC 792 – Internet Control Message Protocol." Available at http://www.ietf.org/rfc/rfc792.txt, September 1981 [April 26, 2007].

[Rowstron2001] A. Rowstron, and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany: 329-350, Nov 2001.

[Rowstron2001] A. Rowstron, and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[Salvador2005] P. Salvador, and R. Valadas, "A Network Monitoring System with a Peer-to-Peer Architecture," in *3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurements*, 14-15 March 2005, Warsaw, Poland.

[Shavitt2005] Y. Shavitt, and E. Shir, "DIMES: Let the internet measure itself," in *SIGCOMM Computer Communication Review*, 35(5): 71-74, 2005.

[Simpson2004] C. R. Simpson Jr., and G. F. Riley, "NETI@home: A distributed approach to collecting end-to-end networks performance measurements," in *Proc. of Passive & Active Measurement (PAM)*, Antibes Juan-les-Pins, France, April 2004.

[Srinivasan2002] S. Srinivasan, and E. Zegura, "Network measurement as a cooperative enterprise," in *Lecture Notes In Computer Science*, vol. 2429, pp. 166–177, March 2002.

[Srinivasan2003] S. Srinivasan, and E. Zegura, "M-coop:A Scalable Infrastructure for Network Measurement," in *Third IEEE Workshop on Internet Applications (WIAPP '03)*, San Jose, CA, June 2003.

[Stoica2001] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *SIGCOMM'01*, San Diego, California, USA, August 27-31, 2001.

[Veiga2004] H. Veiga, T. Pinho, J. Oliveira, R. Valadas, P. Salvador, and A. Nogueira, "Active Traffic Monitoring for Heterogeneous Environments," in *Proc. of 4th International Conference on Networking, ICN 2005*, Reunion Island, France, April 17-21, 2005, Lecture Notes in Computer Science, Springer-Verlag, Vol. 3420 / 2004, ISBN 3-540-25339-4, pp. 603-610.

[Veiga2006] H. Veiga, R. Valadas, P. Salvador, A. Nogueira, T. Pfeiffenberger, and F. Strohmeier, "OWAMP Performance and Interoperability Tests," in *Proc. of 4th International Workshop on Internet Performance, Simulation, Monitoring and Measurement, IPS-MoMe 2006*, Salzburg, Austria, February 27-28, 2006.

[Wen2007] Z. Wen, S. Triukose, and M. Rabinovich, "Facilitating Focused Internet Measurements," in *ACM SIGMETRICS*, 2007.

[Wikipedia] Wikipedia, "Gnutella." Available at http://en.wikipedia.org/wiki/Gnutella, April 24, 2007 [April 26, 2007].

[Zhao2004] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," in *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, January 2004.

Applications

[Azureus] "Azureus", Available at http://azureus.sourceforge.net/ [April 26, 2007].

[BearShare] "BearShare", Available at http://www.bearshare.com/ [April 26, 2007].

[BitComet] "BitComet", Available at http://www.bitcomet.com/ [April 26, 2007].

[BitTorrent] "BitTorrent", Available at http://www.bittorrent.com/ [April 26, 2007].

[DipZoom] "DipZoom", Available at http://dipzoom.case.edu/ [April 26, 2007].

[Eclipse] "Eclipse", Available at http://www.eclipse.org/ [April 26, 2007].

[eMule] "eMule", Available at http://sourceforge.net/projects/emule/ [April 26, 2007].

[Ethereal] "Ethereal – A Network Protocol Analyzer", Available at http://www.ethereal.com/ [April 26, 2007].

[FlashGet] "FlashGet", Available at http://www.flashget.com/index_en.htm [May 17, 2008].

[Gnucleus] "Gnucleus", Available at http://www.gnucleus.com/Gnucleus/ [April 26, 2007].

[JOWAMP] "J-OWAMP", Available at http://www.av.it.pt/jowamp/ [April 26, 2007].

[Kazaa] "Kazaa", Available at http://www.kazaa.com/ [April 26, 2007].

[KTorrent] "KTorrent", Available at http://ktorrent.org/ [April 26, 2007].

[LimeWire] "LimeWire", Available at http://www.limewire.com/ [April 26, 2007].

[MRTG] "MRTG – Multi Router Traffic Grapher", Available at http://mrtg.hdl.com/ [April 26, 2007].

[NeTraMet] "NeTraMet", Available at http://www.caida.org/tools/measurement/netramet/ [April 26, 2007].

[NTOP] "NTOP – Network TOP", Available at http://www.ntop.org/ [April 26, 2007].

[OWAMP] "Internet2 OWAMP", Available at http://e2epi.internet2.edu/owamp/ [April 26, 2007].

[Shareaza] "Shareaza", Available at http://www.shareaza.com/ [April 26, 2007].

[TCPdump] "tcpdump", Available at http://www.tcpdump.org/ [April 26, 2007].

[µTorrent] “µTorrent”, Available at http://www.utorrent.com/ [April 26, 2007].

Others

[Ark] “Ark” Archipelago Measurement Infrastructure. Available at http://www.caida.org/projects/ark/ [May 2008].

[DIMES] The DIMES project. Available at http://www.netdimes.org/new/ [May 2008]

[NETI@home] NETI@home. Available at http://www.neti.gatech.edu/ [May 2008]

[NMT] “Network Monitoring Tools.” Available at http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html, April 26, 2007 [April 26, 2007].

[Nullsoft] “Nullsoft”, Available at http://www.nullsoft.com, [April 26, 2007].

[PlanetLab] “PlanetLab”, Available at http://www.planet-lab.org/ [May 2008].