PERIODICITY OF SNMP TRAFFIC

Bachelor of Science Thesis

Author: J.G. van den Broek

Date: August 1st, 2007

Committee: dr. ir. A. Pras
dr. ir. P.T. de Boer
Prof. Dr. J. Schönwälder

Our problems are man-made, therefore they may be solved by man.
No problem of human destiny is beyond human beings.
- John F. Kennedy

Abstract

The Simple Network Management Protocol (SNMP) is currently widely in use to control, monitor, and configure network elements. Even though SNMP technology is well documented and has been the topic of research a number of times in the past, not much research has yet been done with respect to periodicity in SNMP traffic.

This thesis aims at the development of a method through which periodic and aperiodic SNMP traffic can be separated. This will be achieved by the creation of a set of algorithms which in all are capable of finding relations between single SNMP messages, resulting in sets of related SNMP messages. These sets will subsequently be marked as either periodic, or aperiodic.

The developed algorithms are implemented in a toolset, which can be used for the analysis of available SNMP trace files. The analysis results suggest that the developed algorithms are capable of separating the two SNMP traffic types in nearly all cases. The toolset and the acquired results of this research can be used as a basis for possible future research regarding just either of the two SNMP traffic types.

Acknowledgements

My first three years of studying at the University of Twente ended with a Bachelor of Science assignment, which resulted in this thesis. After almost three years of following courses, I received this assignment in which I had to apply the obtained knowledge and the developed vision. This assignment has been a real challenge for me and I could not have done it without the help and support of a number of other people.

I would like to express my deepest appreciation to my assignment supervisor dr. ir. A. Pras, who works at the department of DACS at the EWI faculty of the University of Twente, for his critical view, constructive level of criticism and the time he has made available for our regular meetings. I also would like to thank him for giving me the opportunity to present my intermediate findings of this research at the EMANICS WP7 meeting in Bremen, Germany.

Secondly, I would also like to thank the other committee members for their constructive input during the course of my research period: dr. ir. P.T. de Boer (DACS department) and Prof. Dr. J. Schönwälder (Jacobs University).

Furthermore, a special thanks goes out to Mrs. Chapman (MA MALT, BEd Hons), for suggesting some language related changes to the final draft version of this thesis.

Finally, I would like to thank my family for expressing their continuous support and words of encouragement.

Gijs van den Broek

This page intentionally left blank.

Contents

Chapter 1

Introduction

1.1 Motivation

The Simple Network Management Protocol (SNMP) is currently widely in use to control, monitor, and configure network elements including: network switches, printers, backup power supplies and many other types of elements. Even though SNMP technology is well documented and has been the topic of research a number of times in the past, not much research has yet been done with respect to periodicity in SNMP traffic.

The desire for this topic of research developed during a time where large SNMP trace files, containing possibly millions of recorded SNMP messages from a particular network, are available to researchers. These files contain SNMP messages that occurred during a certain period of time on a single network. In the long term, it is expected and desirable that specific research will take place on only either aperiodic of periodic SNMP traffic that exists in these trace files. This marks the need for a method which can split these trace files into two categories. After a separation method is developed, subsequent research can take place on either of these two traffic categories.

The research regarding the topic at hand may be approached via a number of ways, but most notable via one of the following three: through the use of Fourier analysis, the detection of traffic patterns of well known SNMP applications, or the mapping and relating of SNMP messages to one another by extensively analyzing the SNMP protocol characteristics. The first one would involve the sole use of Fourier analysis, which may not yield the desired results concerning this topic, because there may be loss of traffic type identification capabilities. The second suggestion would involve research of every implementation specific characteristic, possibly through reverse engineering of the standard SNMP agent and manager implementations that are in use today. Using this knowledge of patterns, it may be possible to remove all periodic traffic (e.g. polling traffic) from the traces, which would remain the aperiodic portion. The third option is the one that is chosen for this thesis. It is expected that through this more generally applicable approach, it will be possible to identify portions of the trace files that are periodic or aperiodic in their occurrence.

In order to create a basis for the stated possible future research related to specific forms of either periodic or aperiodic SNMP traffic, research on the topic of separating these kinds of traffic from each other is paramount. It is therefore the main aim of this thesis to develop methods and algorithms necessary to accomplish this form of separation in the available SNMP trace files and thereby paving the way for future research.

1.2 Problem description

The above explanation gives an abstract view of what is to be achieved in this thesis. But, in order to give a better explanation of what the stated problem exactly encompasses, one must first understand the interpretation of the term ‘traffic’ within the context of periodic and aperiodic SNMP traffic. SNMP traffic, as such, may range from a single request-response sequence, to a complete table lookup. But, which interpretation of the stated term is the right one within the context of this thesis? Following are two scenarios of real world possibilities that stress the importance of understanding this definition of traffic and do so by showing possible relations between SNMP messages:

- Chapter 1.2.1 describes a scenario with a single SNMP manager and a single SNMP agent. This scenario only considers periodic SNMP traffic, occurring in regular polling instances;
- Chapter 1.2.2 describes an extended version of the previous one, in which one manager polls multiple agents. This scenario describes both a case in which periodic polling occurs and one in which that does not occur;

- Chapter 1.2.3 summarizes the whole problem, based on the two preceding scenarios.

1.2.1 Single manager and single agent

This first scenario discusses the first few possibilities of traffic relations through the elaboration on a setting involving only a single SNMP manager and a single SNMP agent. The following examples will stress the importance and difficulties of identifying relationships between any two or more SNMP messages. This scenario will be used for the following cases:

- A basic case involving a single agent and a single manager which is only interested in the contents of a single column of a table available on that agent;
- The same manager and agent, but the manager is in this case also interested in the contents of another column of a table available on that agent.

Consider a simple network scenario, consisting of a single workstation connected, via a LAN, to a network printer. Take into account that the workstation has some SNMP management software installed that regularly polls some values on the network printer, which has the roll of an SNMP agent. The following figure gives a brief overview of this situation.

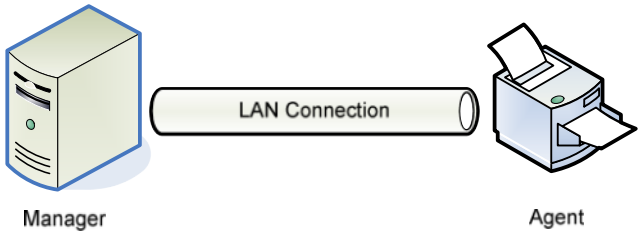

Figure 1.1: Scenario visualization

The SNMP agent module on the network printer maintains a couple of values in three different single-column tables. These tables have the following OIDs: α, β and γ. Each of these tables has a variable number of listed values. Any of these values can be requested by an SNMP manager through the use of one or more get, get-next or get-bulk requests.

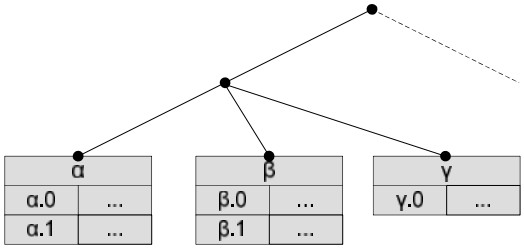

Figure 1.2: Lexicographical order of available single-column tables at the agent side

The management station may be set up so that it will retrieve all the values from α-table every 300 seconds. It is not interested in any value that is not part of this α-table.

Following is a time sequence diagram describing the flow of SNMP messages that include the actual requests made by the manager and the subsequent responses given by the network printer.

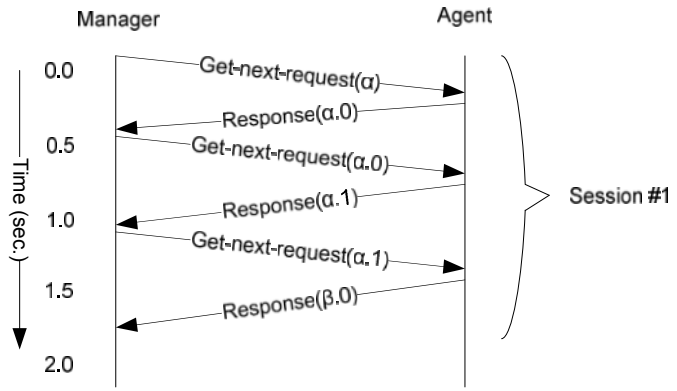

Figure 1.3: TSD showing a single polling instance of the α-table

This time sequence diagram shows the manager sending a get-next-request, asking the agent for the first value of the table that is referenced by the OID α. The agent answers with a response, containing the value of the first instance of the α-table. The management application apparently continues asking for the value that lexicographically follows the one that was just given by the agent. The agent replies with the corresponding value, the second value in the table. The manager carries on and sends its third request: requesting the value that is lexicographically following α.1. This may be the next value of the α-table or, if that value does not exist, the first value of the table following the α-table. Because α.2 does not exist, the agent replies with the value of the first item in the β-table (β.0). The manager interprets the received OID as the endpoint of the single-column α-table.

The described scenario makes it clear that it is difficult to determine which SNMP traffic is to be considered in the process of determining periodicity in SNMP traffic. The given scenario consists of a couple of requests and subsequent responses that all seem intertwined when one looks at the listed OID in each these messages respectively. So, should individual SNMP messages be considered periodic or aperiodic, or should this whole given set of SNMP messages be considered periodic, if this α-table retrieval process happens on a regular basis?

The first one is least likely to be considered correct, since a manager will always reference a single scalar or request the contents of a table or possibly multiple tables in the case of polling. Therefore, under normal conditions, there are always messages related to one another. Taking another look at the given time sequence diagram and without having any knowledge of possible future patterns, the question arises: which SNMP messages belong to each other? After considering the operational behaviour characteristics stated earlier, one might suggest that the shown three requests and three responses are related to each other, since they are all involved in a single table retrieval process. This relation will be called a *session*, which will described further in chapter 2.

The request of the contents of multiple tables is also possible, as well as only a single table. Consider the present scenario and take into account that the given manager is also interested in the contents of the single-column γ-table, as shown in figure 1.2. So, the manager is now interested in the contents of both the α-table, as well as the γ-table on a periodic basis. But, since the data type of the γ-table is such that it will always contain only one scalar, the manager only asks for this single scalar. A polling instance of this extended scenario is shown in the following time sequence diagram.

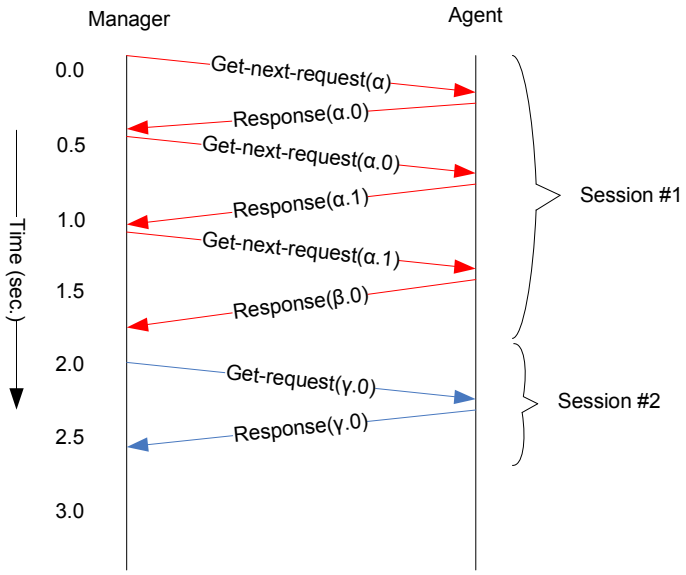

Figure 1.4: A TSD of an extended scenario

The first part of this time sequence diagram is the same as in figure 1.3. However, after completing the retrieval of the values in the α-table, a get-request, asking for the first value of the γ-table is shown. By not asking for the second value of the single-column β-table, it becomes clear that the manager is not interested in the values belonging to that particular table. This is in line with the settings of the SNMP management application, as described earlier. Since the SNMP management application is aware that the OID belonging to γ only encompasses a single scalar, it will not ask for any other values of that table, as can be seen in the TSD.

As has been shown in figure 1.3, the first three get-next-requests and their related respective responses show a clear relationship. It is apparent that through the sending of these three requests, the content of a single table is retrieved. The fourth request is a get-request for a value in a completely different table. Therefore, in this case, two different and independent sessions may be identified. The second session and its messages are coloured blue in figure 1.4.

The above examples are simple ones. The identification of sessions in general is significantly more difficult and will therefore be discussed more thoroughly in the course of this thesis. These examples have only stressed the significance of identifying possible relationships between the occurrences of SNMP messages.

1.2.2 Single manager and multiple agents

Two possibilities of related SNMP traffic have been shown so far. This second scenario discusses the possibility that the stating of relationships in the form of sessions alone may not be enough. It may be possible that sessions are a part of a larger set of sessions that are all related in their occurrence. The following examples will stress the importance and difficulties of identifying such relationships between any two or more sessions. This scenario will be used for the following cases:

- An extended version of the one of the previous subsection, but now involving multiple agents that are being polled on a periodic basis;
- The same manager and agents, but with the addition of aperiodic polling.

Consider the network setting of the previous example. In this example that scenario will be extended with two other network elements that operate as agents. A single manager will regularly poll not only the network printer, but also a network switch and a workstation. In total, this scenario includes a single management workstation and three different agents. The following figure shows the involved connected elements.

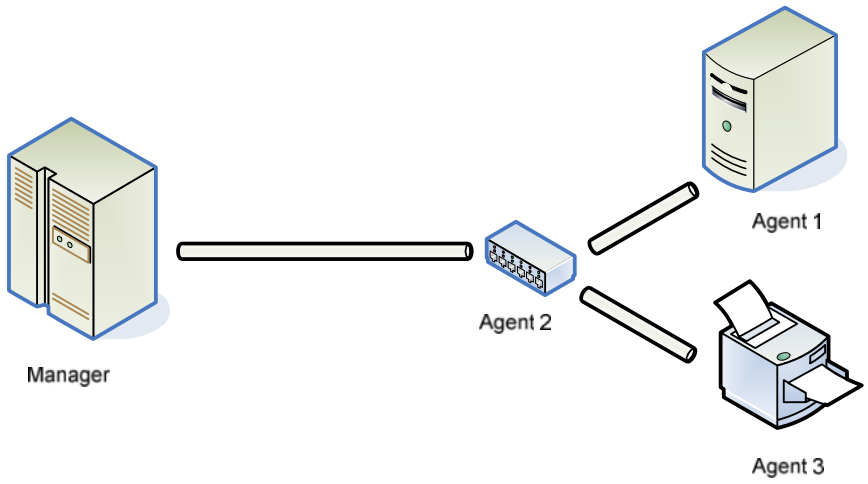

Figure 1.5: Scenario visualization

The available information, that is stored in various tables at the agents, remains unchanged for the case of the network printer with respect to the tables in the previous example. Besides that, both agent 1 and agent 2 contain numerous tables of information that can be queried. However, the manager is only interested in the complete contents of the single-column ε-table in the case of agent 1, the single-column γ-table in the case of agent 2 and only the single-column α-table from agent 3. The manager is set up to retrieve all the values from these selected tables every 300 seconds. Following is a time sequence diagram describing a single polling instance, in which all of the three agents are being queried for the contents of the selected tables.

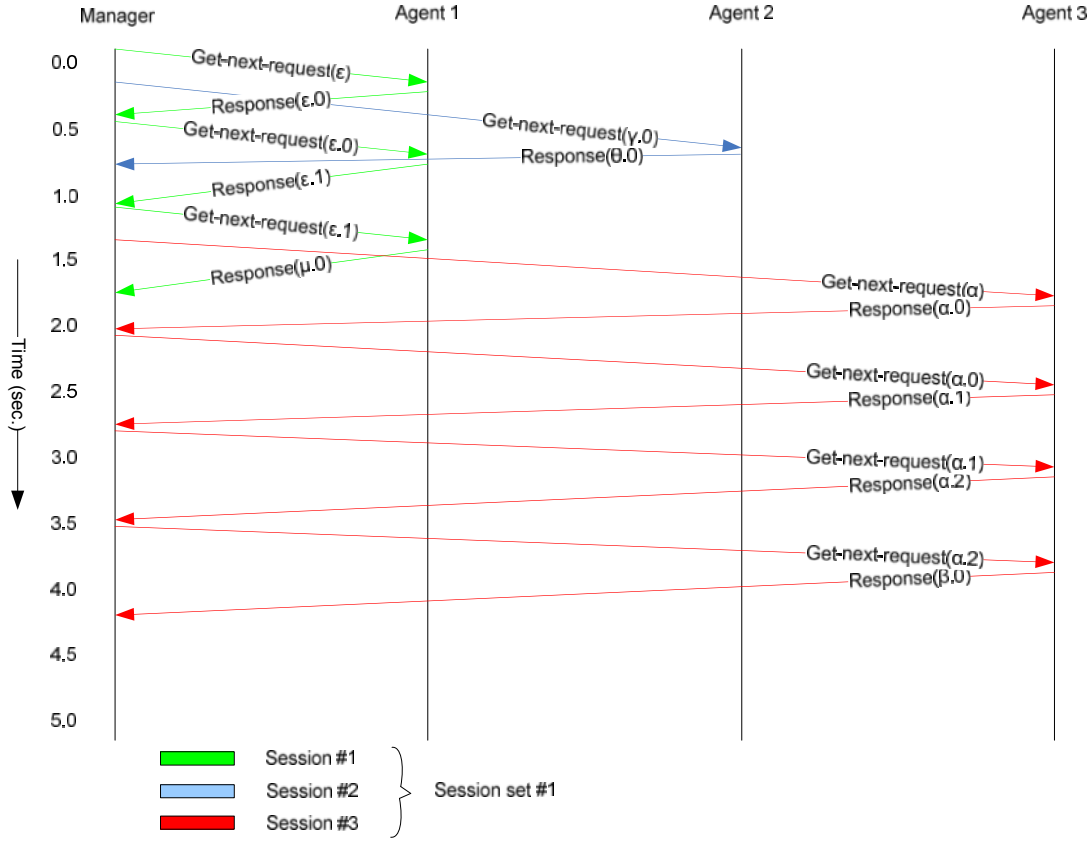

Figure 1.6: A single set of sessions involving one manager and three agents

This TSD shows that at the beginning of the diagram the values in the ε-table are being consulted at agent 1. A few milliseconds later, the first get-next-request is sent to the second agent. Since the γ-table apparently does not contain any values, a response is returned with the first value of a completely different table, of which the manager does not have any interest in. Again a few moments later, still while the ε-table is being read on agent 1, the values in the α-table of agent 3 are being requested. A little past 4 seconds is eventually needed to retrieve all the desired values from the three agents.

Another interesting aspect of this TSD is the fact that sessions, involving different agents, can take place simultaneously or almost simultaneously. This is one of many profound real world characteristics that must be considered with regard to this thesis problem. One of the consequences of this phenomenon is that sessions occurring between a manager and an agent may not always occur in the same order. Take again in mind that the manager is set to poll the three agents every 300 seconds. The following graph shows a possible scenario, describing three subsequent polling instances.

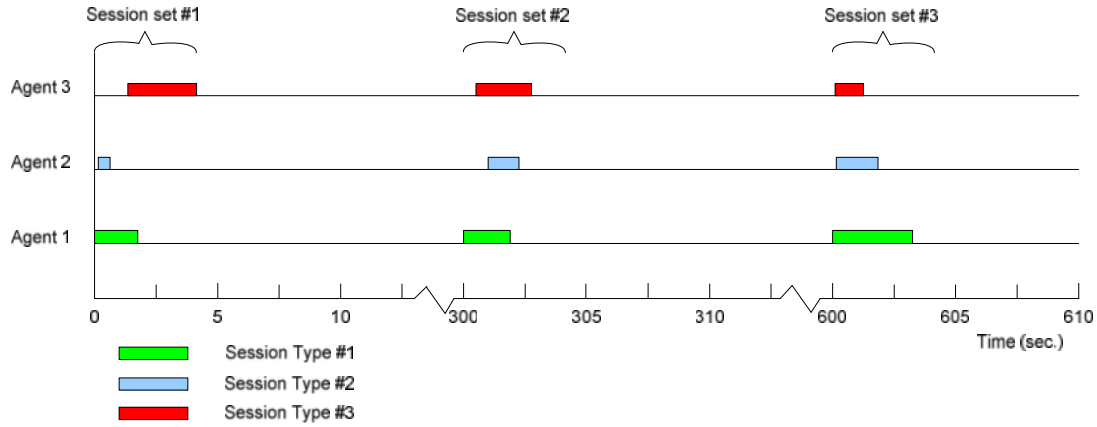

Figure 1.7: Sessions involved in regular polling instances of the three different agents

This scenario depiction shows a possible execution of three polling instances, where every bar represents a period, during which SNMP messages flow between the manager and the respective agent. The first shown polling instance is the same as in the given TSD, shown in figure 1.6. The other two show the possibilities of a table increasing/decreasing in length over the course of a few minutes and the difference in the order of polling by the manager of the three agents. These are just a few of the many characteristics that have to be taken into account for this research topic, since these are all real world possibilities.

This view also suggests a clear relationship between the sessions that occur within a single polling instance, or *session set*, which is discussed thoroughly in chapter 4. Also, each of these sessions is of a certain *session type*. For example, a session that occurs between the manager and agent 1 is of a specific session type and a session between the manager and agent 2 is of a different session type. A more elaborate discussion on the exact purpose and definition of a session type is discussed in chapter 3. It also appears that every session type occurs once in every session set. This is no surprise, because it was already known what the manager settings were beforehand. However, this information about a manager is not likely known in the case of the trace files that are to be used for this research. In order to still be able to identify possible relationships between sessions, the question is posed: how to determine which sessions are related and form session sets?

The above scenario describes a 300 seconds interval between every start point of a session set. As is shown, this gives a clear periodic pattern. But, not every real world case will be as clear as that. The following graph shows the occurrences of the different sessions in case the stated manager is set to poll only the selected agents at times on which the user of the manager application desires the information of the mentioned tables.

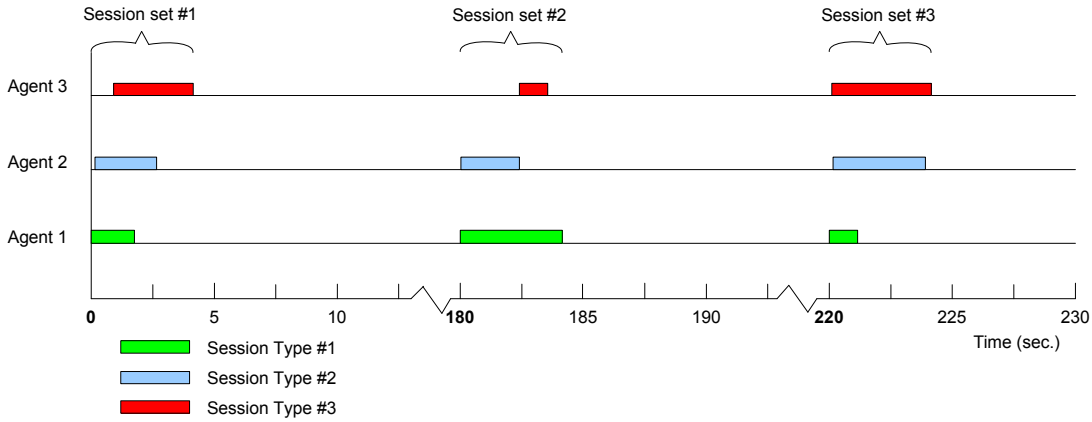

Figure 1.8: Irregular polling instances

Again, it becomes apparent that certain sessions are related and form session sets. However, these session sets do not occur in a regular fashion based on a certain interval. It is therefore that the following question is raised: once a couple of session sets have been identified, when are they considered periodic and when aperiodic?

Another question that would apply is about the finding of intervals in periodic traffic. In the first scenario of this subsection, describing periodic behaviour in the form of regular polling, it can easily be seen what the interval is. However, there may be scenarios in which it may not easily be seen which intervals there are.

1.2.3 Problem summary

This subsection has stressed the difficulties concerning the term 'traffic' within the context of the search for a method that is capable of splitting SNMP traffic in periodic and aperiodic traffic. In this subsection have three possibilities of SNMP traffic relations been discussed briefly. The scenario in chapter 1.2.1 clearly shows the general difficulty of identifying relations between single SNMP messages. Still, the possibility that single SNMP messages should be considered as either periodic or aperiodic seems certainly incorrect. What remains are the two other possibilities. The discussion in chapter 1.2.2 shows that sessions alone may also not be sufficient, since sessions are likely to be related to the occurrence of other sessions. Thus, sessions could very well be part of a larger session set containing one or more related sessions.

Moreover, out of these session sets can become even larger relationships, describing sets of session sets, which could be used for the determination of periodicity. But, in order for a specific session set to be considered part of a set of session sets, all of its sessions must be of a session type that can also be found in the other session sets that are of that same set of session sets. In other words, there is a necessity to be able to compare different sessions. This raises the last question: how to determine the session type of a certain session?

1.3 Research questions

The central theme of this thesis is the separation of periodic and aperiodic SNMP traffic from each other. The previous section already laid out some of the problems concerning the identification of relations within SNMP traffic. Still, there are two ways via which periodic and aperiodic SNMP traffic can be separated from each other. One of them is by identifying periodic traffic and removing that, leaving the aperiodic traffic, or by identifying the aperiodic part, leaving the periodic part. Since the latter option is more difficult to achieve, because there is no specific expectable pattern in aperiodic traffic, the first option is chosen.

By identifying the periodic SNMP traffic, the remaining part is automatically the aperiodic part. But, this identification process is characterized by quite a number of problems, as has already been shown in section 1.2. First, relations in the form of sessions need to be found, then possible larger relations, in the form of session sets are to be identified, which combined with other comparable session sets may be marked as either periodic or aperiodic. But in order to do so, one must have the ability to determine a session type of a session, in order to compare the sessions making up a session set. This is to be used to compare session sets for 'equality'. After that needs to be determined whether a certain set of related session sets have a specific periodic or aperiodic behaviour. If this would be periodic, it is also important to know which intervals can be found in that specific periodic set of session sets.

Clearly, a number of steps need to be made in this process. For this reason is the main thesis problem divided into smaller sub problems that are stated here. The combined answer of each of these questions will yield the solution to the main thesis problem.

How to determine which SNMP messages belong to a single session?

The examples in the previous subsection show that in order to be able to identify 'traffic' as either periodic or aperiodic, one first needs to be able to identify relationships between SNMP messages. This question encompasses the identification of SNMP messages that contribute to a single session.

How to determine the session type of a session?

After sessions have been identified, it needs to be known of which session type a certain session is. This way, different sessions can be compared for 'equality' and this forms in turn a basis for the next step: the identification of larger forms of relations in the form of session sets.

How to determine which sessions form session sets?

Once the basic relationship between SNMP messages can be made, a method needs to be found to identify any possible larger sets of sessions that could be related in their occurrence. The answer to this question will form a basis for the next step: the determination of which session sets are to be marked periodic and which aperiodic.

When is a session set considered (a)periodic?

Once related session sets can be identified, the next step is the determination of whether or not this set of sessions and all their respective SNMP messages that make up these sessions, is to be considered periodic or aperiodic.

How can intervals in periodic SNMP traffic be determined?

Only after a separation of traffic can take place will it be possible that either of these two types of traffic can be analysed further. One of the most significant characteristics of the periodic traffic part will be the intervals that are part of the periodic behaviour. Since multiple intervals might exist within a portion of periodic traffic, a method needs to be determined that is able to find all the intervals that exist in that set of traffic. The research regarding this question will also include an overview of intervals found in actual SNMP trace files.

1.4 Approach and thesis outline

The approach of this thesis will be to position it around the already numerous existing SNMP trace files. The eventual goal is to split these trace files into a category of SNMP messages that contribute to the periodic part and another category which consists of the SNMP messages contributing only to the aperiodic part.

This goal clarifies the need for algorithms that, when put together, yield the desired result. These algorithms, which will be discussed extensively in chapters 2 through 5, will include one that is capable of finding basic relations between single SNMP messages, resulting in sessions. A second algorithm will be discussed in chapter 3 that can differentiate between different sessions. Thirdly, there will also be the need for an algorithm that identifies inter-session relationships, which should result in the capability of finding related sessions that would form session sets. It is then that an algorithm will be required that can state whether or not a certain session set is periodic or aperiodic in its behaviour. This same algorithm will also have to determine the intervals that can be found in periodic session sets.

Since the actual application of any developed algorithm requires the translation to computer programming code, it is also important to find out what needs to be programmed and which computer applications are already available for this phase in the research process. As will also be explained later on, no complete application is yet available, however some existing applications can be used as a basis [1]. It is for this reason that a toolset needs to be written to fulfil and automate all the tasks that make up the larger set of algorithms that are to be developed. The description of the complete implementation is given in chapter 6.

After the algorithms have been developed and implemented, the combined toolset needs to be applied on the available SNMP traces. The results of the total toolset of computer programs, as well as intermediate results from the various members of the toolset, which are all discussed in chapter 7, will all in all help to find the answers to the listed research questions, as well as back up statements made throughout this thesis.

This thesis will end with a concluding chapter, chapter 8, which contains an overview of the stated research questions and their respective answers. Some recommendations for future research will also be given in that same chapter.

1.5 Intended audience

This thesis is written for an audience consisting of those that are generally interested in SNMP related research and researchers that operate in the field of SNMP, especially those that are interested in periodicity of SNMP traffic. It is expected that this audience has a significant level of understanding of the general operation of SNMP managers and agents. Readers are also expected to have some knowledge of the SNMP protocol.

As a result of the above expectations, no elaboration will take place in this thesis on the general way of operation of the SNMP protocol, nor that of SNMP managers and/or agents. However, whenever there could be any ambiguity on the meaning or operation of anything related to any of the above, a short description will be given.

1.6 Related work

During the time of writing, already multiple researches have been carried out at the Jacobs University (Bremen, Germany) that have had some relation to the analysis and/or determination of periodicity of SNMP traffic. For instance, a more general discussion regarding this topic can be found in [2]. Besides that, a more recent MSc thesis by C. Ciocov [3] touches the topic in hand, though it discusses this only briefly and poses no solutions to the problems raised in this thesis. A more elaborate and detailed approach towards solving the central problem raised in this thesis was also attempted by M. Harvan [4]. He has made some significant steps in this field of research. However, his approach - which makes extensive use of Fourier analysis – did not yield a complete and proper identification of intervals within periodic SNMP

traffic, nor did he address the issue of separating the two types of SNMP traffic extensively. Still, his work can be considered as significant within the context of this research topic.

At the University of Twente, I. Grondman [5] has done some research in the recent past, concerning periodicity of internet traffic in general. With his research, with respect to SNMP traffic, he only considered the periodicity of SNMP traffic as a whole. He did not look at periodicity at a more detailed level, nor did his research include the separation of periodic and aperiodic SNMP traffic.

Clearly, plenty of research is either currently taking place, or has taken place in the past year that to some extent could be related to the topic of this thesis. Still, none of the mentioned research projects take a thorough and in-depth approach towards the separation of the two types of SNMP traffic. It is therefore the aim of this thesis to bring the current level of research on this topic to a new level, by taking steps that have not been taken before.

Chapter 2

Session determination

At this point it has been made clear what the general approach is that has been chosen to solve the problems raised in this thesis. Furthermore, some information has already been given regarding the steps that have to be taken as part of the stated solution. Still, it is unclear what a session exactly is and how it can be detected. It is the purpose of this chapter to answer these questions.

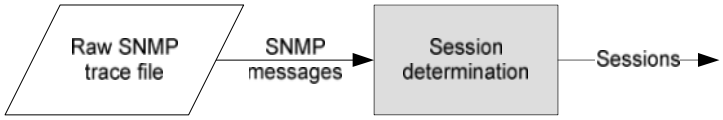

Figure 2.1: Input and output of this algorithm

The first step, as part of the chosen solution, is the discovery of sessions, based on raw input. This input is supposed to contain only SNMP messages that can be considered independently. It is also assumed that this input consists of SNMP messages that have been recorded on a specific site for a specific duration. As a result, it is expected that this input may be considered complete and therefore does not contain any wholes (i.e. all SNMP messages that could have been recorded on that specific location have been recorded and no SNMP messages have been filtered out prematurely). The output of this algorithm is supposed to be nothing but sessions, such that every message that exists in the recording and was therefore fed to the algorithm as input, also exists in one and only one session in the output.

It is the sole purpose of this algorithm, which has the responsibility of detecting sessions, to detect all sessions possible. It is supposed to detect the most basic form of relationships between SNMP messages that belong together. In chapter 1.2 it has already been shown that the retrieval of values from a single single-column-table and the messages that take part in that respective process, are to be considered as a single session. Also it has been shown that the retrieval of values from *n* single-column-tables, which do not follow each other directly lexicographically, ought not to be considered as a single session. Instead, *n* different sessions should be detected.

But these are only a few cases where sessions can be clearly and easily identified. Real world scenarios, as will be discussed later on, show that there are numerous cases where this is not so easy. Before these scenarios are discussed, first a basic definition of a session is given. Then the most noteworthy scenarios are being discussed with the purpose of extending the given basic definition of a session. After that the complete definition, as well as the algorithm itself, shall be described thoroughly. In brief, the following subjects will be discussed:

- A basic definition of a session is given, based on the information that has been discussed regarding this term so far;
- Some aspects of (likely) real world scenarios that are, or should be taken into account in order to be able find all sessions properly. This will involve scenarios in which, for example, retransmissions occur, or certain columns exist that contain gaps;
- The complete definition of a session, based on the basic variant, as well as the considerations that have been discussed in chapter 2.2;
- The algorithm description, describing all the steps necessary to find all sessions possible in every likely scenario. This algorithm description is based on the complete definition of a session, as well as the issues that have been considered in the sub chapters before that.

2.1 Basic definition

The term session has been used in chapter 1.2 only briefly for introductory purposes. But, the usage of this term, as well as the depiction of sessions in many diagrams, already gives a good impression of the meaning of a session. Based on this information, the following observations can be made about a session:

- It involves one or more SNMP messages;
- The involved SNMP messages are exchanged between exactly two network elements;
- All messages, minus the responses, are of the same operation type;
- The OID in every SNMP message appears to lexicographically increase, compared to the OID in the chronologically previous SNMP message of that session.

Combining these four points into a basic definition of a session yields:

Definition A *session* is a set consisting of one or more SNMP message, which are all exchanged between exactly two defined network elements. Furthermore:

- The messages in a session other than responses all have the same operation type;
- The OID in every message in a session is either lexicographically the same or increasing compared to the OID in the chronologically previous message within a session.

As a result, sessions will consist of SNMP messages that overall at least adhere to this definition.

2.2 Considerations

The previous subsection discussed a very basic definition of a session. But this depiction is not ready to cover most of the real world problem scenarios like: what if requests contain multiple OIDs, or what about SNMP messages which are not requests at all? The most significant cases are highlighted here, which in turn will assist in the determination of the complete definition of a session, as well as the algorithm itself.

It should be noted that all of the following subsections discuss cases that may not be considered as sessions, based on the current definition of a session, but which are desired to be considered as such. In order to keep the scenarios and examples uniform in their basics, the following information about the involved manager and agent shall be used in the subsequent scenarios.

The agent that is involved in the scenarios has a number of single-column tables available, consisting of one or more values that can be referenced by a manager. The following tree view shows some of the available tables on the agent side.

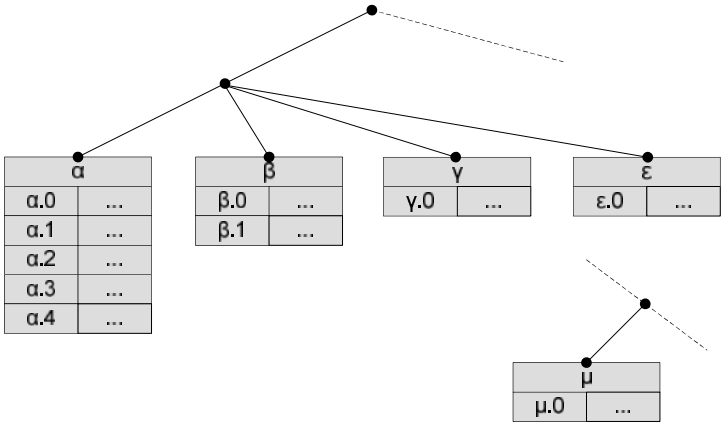

Figure 2.2: Tree view of available tables at the agent side

This tree view shows some of the tables that can be referenced by a manager on a certain agent. It shows one table with five values, one with two values and three tables with only one value. It should also be understood that the tables α, β, γ and ε come lexicographically before table μ.

2.2.1 Multiple referenced OIDs

So far in this thesis, only simple get and get-next operations have been described within the context of sessions. All of these only consist of request and subsequent response messages that contain only one referenced MIB object. However, there are plenty of cases in practice where more than one MIB object is referenced.

To signify this, consider a scenario involving the single manager and single agent mentioned. Now, assume that a manager is set to retrieve the contents of only table α. It is not interested in any value of table β, γ or ε. Since it may be the case that the table contains a significantly large number of values, the manager application chooses to use get-bulk requests to retrieve all the values. These get-bulk requests also contain a request for the first item of the μ-table, which will be requested with every request as a non-repeater. The following time sequence diagram shows a session in which the selected values are retrieved. However, this diagram highlights some issues that may require an extension of the definition of a session.

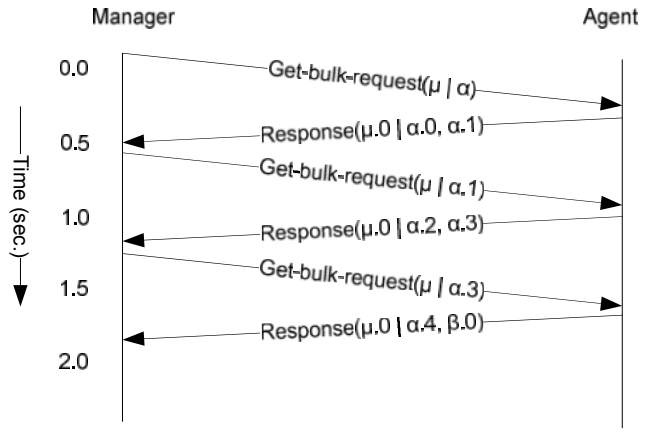

Figure 2.3: TSD describing a possible get-bulk operation

This TSD may seem odd, still it underlines a couple of real world problems that have to be taken into account, before a complete definition of a session can be given. Clearly, there are three get-bulk request messages requesting the desired information from the described agent. But, instead of just one OID being positioned in a request and subsequent response, there are now multiples. Moreover, there is a non-repeater section of the request that does not increase lexicographically compared to either that same position in the previous request or previous response message.

This scenario highlights the following points that have to be taken into account for the completion of the definition, as well as for the construction of the algorithm:

- Request and response messages may contain more than just one OID;
- Some OIDs may not increase lexicographically, compared to the same position in the list of OIDs in the chronologically previous message.

2.2.2 Timeout and retransmissions

Another real world occurrence is the fact that SNMP messages may not arrive at their designated location, as a result of various reasons. This is inherent in the usages of UDP as a transport protocol for SNMP messages. Therefore, in case a request or response fails to arrive in time, some mechanism must be triggered, so that the information cycle is still completed.

The following diagram shows an attempted table retrieval activity, initiated by the manager that has been described before. This manager wants to request values from the given agent. In this case, the manager is only interested in the contents of the β-table.

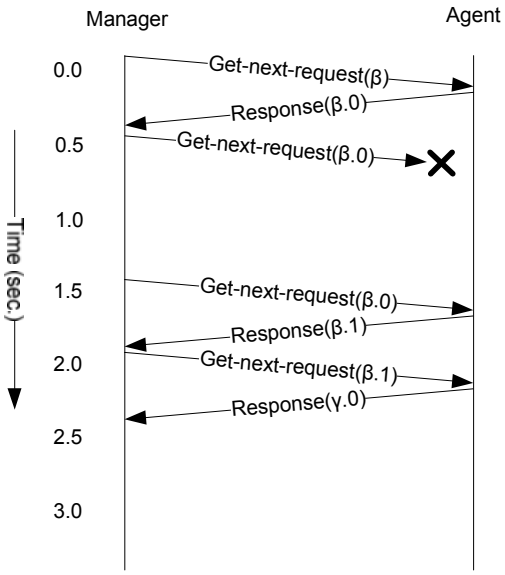

Figure 2.4: TSD showing the occurrence of a timeout and a retransmission

This diagram shows the retrieval of table values, which make up the β-table. However, the second request message fails to arrive at the agent side. A timeout mechanism, operating on the manager side, which is in this case set to resend a request after one second, resends the same request to the selected agent. This second time, the request arrives and is being answered with a response. This goes on until the end of the β-table has been reached.

The most profound characteristic of this scenario is the fact that if a message and its retransmission occur subsequently within a session, without any other messages of that session occurring in between, then none of the OIDs between the two messages may increase lexicographically. This is shown in the diagram with the same OID reference being made in the second and third request (a retransmission of the second request).

Because the SNMP trace files are recorded from a position between a manager and an agent, it is also possible that a retransmission is seen at this recording position after another messages has already been sent. For example, an agent may send two responses, of which one is a retransmission of the first, as a standard measure to reduce the session time in case of a lost response. But, the first response may have reached the manager side and the manager may have sent a new request as a result of that. It may now be possible that this new request is seen at the recording position before the second response is seen. This case would raise the issue that the OIDs compared between this second response and the request occurring just before that may not increase lexicographically. The following diagram shows this possibility. For clarity reasons, the requests and their respective responses (with the same request ID) have the same colour.

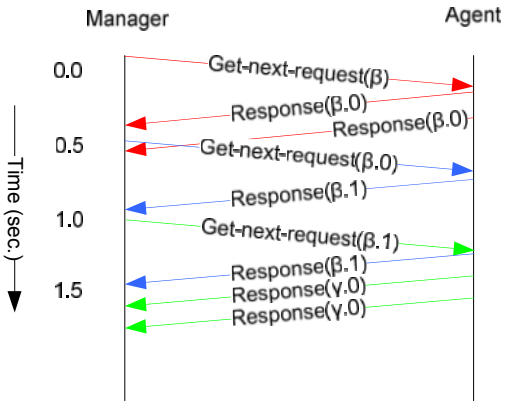

Figure 2.5: Standard retransmissions of responses

It should be noted that a similar situation could occur in the case of a retransmission of a request. In that case, just before a response reaches the manager, a timeout could occur at the side of the manager, which in turn retransmits its last request of which the response is received almost immediately after sending it.

Besides the list of OIDs that should be looked at within the process of extending the definition of a session, one should also look carefully at other fields of an SNMP message. Most notable the request ID field, since a retransmission of a request may have the same request ID, but will have the same list of OIDs compared to the original request. On the other hand, retransmissions of responses will have the same request ID as the request that is being answered [6]. However, the contents of the list of OIDs may be different, because the respective table values may have changed during the time between the first response and its retransmission. Hence, both the list of OIDs and/or their respective values may be different between two or more responses to the same request.

The events of timeouts and subsequent retransmissions show a couple of aspects that have to be taken into account, in order for a session, such as for example the one displayed in figure 2.5, to be properly considered as such:

- 'Equal' requests and response messages may occur two or more times within a single session. Retransmissions of requests *may* have the same request ID and *will* have the same list of OIDs. Retransmissions of responses *will* have the same request ID, as the request that is being answered, but the contents of the list of OIDs and their respective values *may* be different;
- If a retransmission of a message is received at the recording site, after a new message has been seen, then the OIDs compared between the retransmission and the message before that, may in some cases not increase lexicographically. In such a case, a retransmission of an earlier message would be compared with a newer message;
- Although not described here, requests may in some cases not be followed by a related response, as a result of the fact that the network element, which has the agent role, is turned off, or is simply unreachable at the time of the request being sent.

2.2.3 Table characteristics

Tables containing one or more values at the agent side may have different characteristics. A table could contain a few or many values. Normally, a value can be referenced by requesting the n^{th} value from a table, resulting in the corresponding value, if that particular table has that many values. However, there are two significant table characteristics that should be noted:

1. One of these is the possibility that a table, consisting of multiple columns, may have columns that are of unequal length. So, consider for example that the OID α represents the first column of a two-column table and OID β the second column and that the α-column contains more items than the β-column;
2. A second possibility is that one or more columns of a table contain gaps. This means for example that $\alpha.2$ and $\alpha.4$ can be referenced, but $\alpha.3$ not. A get-next request querying $\alpha.2$ would simply result in a response containing $\alpha.4$ (the lexicographically first item following $\alpha.2$).

These two possibilities haven been visualised in the following time sequence diagram, which shows an apparent proper session, consisting of get-next requests and their respective responses.

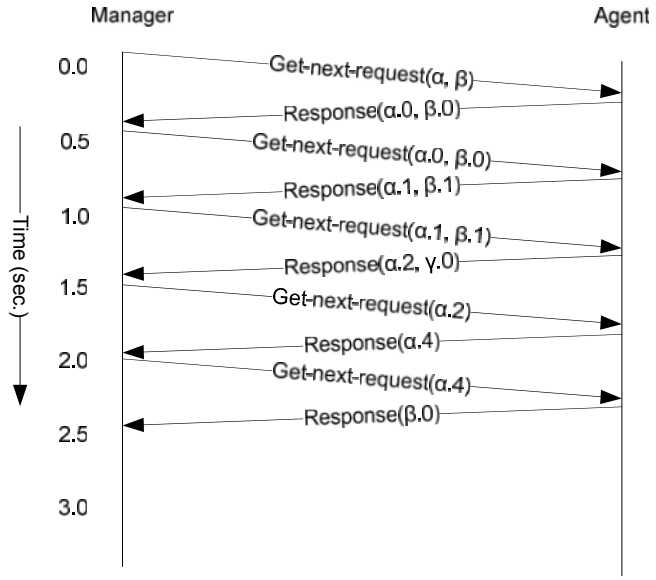

Figure 2.6: TSD describing table gaps and unequal column length

The TSD shows how a manager requests all of the items of two columns of a table at the agent side. Because the β-column contains only two values, the end of that column is reached earlier than that of the α-column. As a result, the manager only carries on with the α-column, which shows the described gap. Still, the manager goes on with the requesting of values from this column, until it notices the end of that column.

These two table characteristics show that the following aspects with respect to these characteristics are important for the definition of a session and the algorithm that will identify the involved messages in a session:

- Not all requests and responses belonging to a single session have an OID list containing a fixed and equal number of items as a result of unequal column length;
- The OID at a certain position in the list of OIDs in a response message may not always match that of the OID on the same position in the respective request as a result of certain table characteristics.

2.2.4 Non-get-like operation types

Up to this point, only get, get-next and get-bulk operation type sessions have been discussed. There are however also a number of other operation types. Although these other operation types will not result in table retrieval processes, as get-next and get-bulk can do, an understanding must still be created regarding the definition of a session with regards to these other operation types. Following are a few time sequence diagrams showing SNMP messages which ought to be considered to be part of the same session respectively. Every TSD shows a single session.

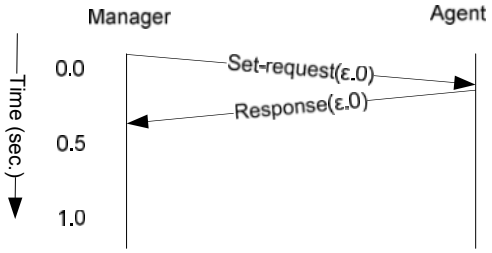

Figure 2.7a: Set session

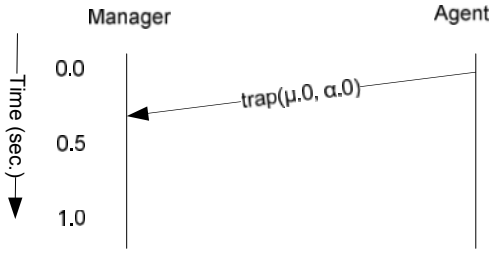

Figure 2.7b: Trap session

Figure 2.7a shows an apparent set operation involving a single set request and a confirming response from the respective agent. Although multiple set operations can take place between a manager and an agent, every single set request occurs on a specific separate basis; multiple set operations ought therefore not to be considered part of a single session. However, also here exists the possibility of retransmissions. Hence this possibility must be considered later on.

About the same can be declared about a trap session, as shown in figure 2.7b. Every occurrence of a trap message is the result of a specific activity, like the rebooting of a certain network element. Therefore, every trap message should be considered as a single session. However, again here applies the exception in the case of a retransmission of a specific trap message.

The following figure 2.7c shows a single inform session, which consists of a request and a response. Since this kind of operation, just like trap and set operations, is raised on an independent basis, should also inform sessions be considered the same way. Also here applies the exception for retransmissions for either or both the request and its response messages.

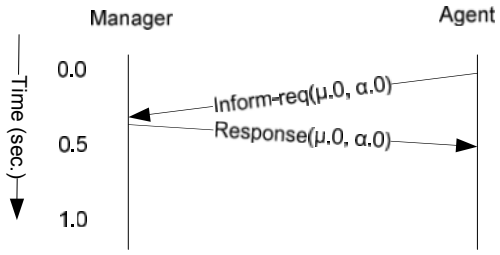

Figure 2.7c: Inform session

What remains is a report session. Since this kind of operation has not been documented strictly, it will be hard to determine how a report session should look like.

Besides the get-next and get-bulk operation types, there are also a couple of other operation types. Although these sessions will be less interesting, due to the fact that there are likely to be only one, two, or maybe more (as a result of retransmissions) messages involved in such a session, these kinds of operations must still be considered. As a result of the stated characteristics for these other operation types, can the following important aspects be highlighted:

- Set and inform sessions should only contain a single request and a single related response, with the exception for retransmissions. In that case, more than one request or response may be assigned to a session, but these must still be equal to an already listed request or response message respectively;
- Trap sessions should consist of only exactly one trap message. However, in case a retransmission is sent of a listed trap message, that retransmission should also be considered part of that specific session;
- The expected behaviour and therefore the relations between report messages and possible related response messages is not strictly documented and leaves room for implementers of SNMP applications to choose how they use this kind of operation. Consequently, the complete definition of a session, as well as the algorithm should be liberal enough, in order to allow for certain kinds of report sessions.

2.2.5 OID insertion and position change

Most scenarios that have to be considered for the proper construction of a definition of a session and the accompanying algorithm have already been considered. There are still a number of aspects that may occur in the real world concerning this topic. These are more implementation specific and have thus nothing to do with protocol related aspects, like the scenarios that have been considered so far. Two of these potentially occurring aspects, related to the placing of OIDs in requests and response messages, will be discussed here.

For this scenario, consider again the manager and the single agent as described before. This time, the manager is interested in the contents of the α-table and the β-table. Consider as well that the manager application is set to request the first item of the μ-table once every three requests. Take also into account that this same manager application is programmed in such a manner, that it may change the order of OIDs being requested, while still being able to retrieve all values from the two tables. The following time sequence diagram shows an example with these two phenomena included. The red requests show the insertion of the additional OID. The green request has a changed order of OIDs.

Figure 2.8: TSD showing the changing of positions of OIDs and the irregular insertion of an extra OID

The diagram shows that the values belonging to the two tables in question are still successfully being retrieved by the manager, despite the change in order in the case of the request marked green. Although this scenario seems odd, still, in the case of bad programming, a manager may still choose to insert a request for the first scalar of the μ-table on a regular basis in a get-next request, instead of choosing for a get-bulk request with the non-repeater field. Another case of bad programming may be the cause of the change in the order of OIDs.

These last two cases may influence the eventual version of the session definition, such that it will allow for these two cases to occur in the available SNMP traces, while sessions are still being recognized properly. The following points should therefore be taken into consideration:

- A manager may request all values of multiple columns through the use one or more sets of requests and responses. However, the order of referenced columns may change along the way, without loss of continuity for the manager. The manager may even choose to retrieve the contents of one column of a table before retrieving the contents of another column of that same table, instead of just walking through all columns of a table at once;
- A manager may be set to request one or more specific scalar values every few requests by adding the desired OIDs to the list of existing OIDs of a request. This addition of OID(s) may happen on an irregular basis.

2.3 Complete session definition

After considering the scenarios listed in the previous subsection, it can be stated that the basic definition, as given in chapter 2.1 needs to be extended, in order to cover all the discussed cases. Taking the points from the basic definition, as well as the points that have been highlighted in every listed scenario, the following complete definition of a session results:

Definition A *session* is a set consisting of one or more SNMP messages, which are all exchanged between exactly two network elements, which are each identified through their IP address and respective port number. Furthermore:

- The messages other than responses in a session all have the same operation type;
- The following applies only to get-next or get-bulk requests:
    - At least one OID in every get-next, or one OID which is not part of the non-repeaters of a get-bulk request, which is also not a retransmission in a session, is lexicographically the same compared to an OID in the chronologically last response to the previous request within a session. This latter OID may be on a different position in the list of OIDs than that of the OID in the list of the request message in question. This request must occur within a specifiable amount of time after the last listed response (if any) and must come from the same network element as the other requests of a session;
- A response message is considered part of a session if it occurred within a specifiable amount of time after an already listed non-response that has the exact same request ID;
- Retransmission will be considered part of a session in the following cases:
    - A retransmission of a response message is considered part of a session if it contains a request ID that is equal to one of the already listed requests in that session and occurred within a specific amount of time after the respective original response message. The list of OIDs and the values may be different;
    - A retransmission of a get, get-next, get-bulk, set, or inform request message is considered part of a session, if it contains a list of OIDs that is equal (though the order may be different) to one of the already listed requests in that session with the same list of OIDs. The request ID may be different;
    - A retransmission of a trap or report message is considered part of a session, if it contains a list of OIDs that is equal (though the order may be different), to an already listed trap or report message respectively;

This very extensive definition of a session may require some additional information. Since a session involves a number of SNMP messages that all occur within a short period of time, it is therefore expected that both the sending party and the receiving party communicate through a single port number and a single IP address respectively. For example, if a manager wants to retrieve the contents of a table located at an agent, it will use a specific port on its side for at least the duration of that session. It would make no sense if a manager would change its port number during the process of retrieving all the values, if this requires multiple requests to be sent, simply because this would make it impossible for retransmissions from the agent, or otherwise delayed response messages, to be delivered properly to the manager application. Also, if the port number would not be used as a filter, the messages coming from and going to different manager applications/threads operating from the same IP address will be inadvertently considered to be coming from/going to the same manager instance. This could result in sessions containing messages that are in fact not related to each other. Also, it is expected that the agent will, as is normally the case, use a fixed port number on which it can be contacted. Usually this is port number 161 [6].

The first point has already been elaborated upon in the previous sections. The second point is a result of many restrictions that have been posed in the various scenario considerations. It is therefore that only this declared link can be made between OIDs of different messages. This OID linkage with the last response also applies only to get-next and get-bulk requests. A get-next and get-bulk request message must occur within a specific amount of time after the last response in order for it to be considered part of an existing session. This OID linkage does not apply to other operation type sessions, because only get-next and get-bulk sessions can contain multiple requests that belong to a single session.

The third option applies a restriction to the addition of response messages to a session. As can be seen, no OID requirements are made here. This is due to the listed table characteristics - like unexpected column endings or gaps - that make the list of OIDs in responses unpredictable. Furthermore, the responses do not say anything about whether or not the manager carries on referencing OIDs based on the last response. Thus, they do not say anything about the manager continuing with a session, or starting a new session.

The fourth option comprises of different cases in which a message is considered a retransmission of an existing SNMP message in a session, in which case that particular retransmission will also be considered part of that same session.

Based on the given definition, the importance of not just looking at the 'normal' get-next and get-bulk sessions, but also at, for example, a trap session should be stressed. No response can ever be found in a trap session, therefore not all sessions will contain responses. As has also already been discussed, even get, get-next and get-bulk sessions may not contain any response in certain scenarios. This could be the case when a manager attempts to request the value of a certain scalar from an agent, which cannot be reached. Nevertheless, a session will always start with an initiating non-response message.

2.4 Algorithm

It is now possible to describe the developed algorithm, now that the complete definition has been given and the most significant problem scenarios have been discussed. As has also been discussed in the introductory part of this chapter, this algorithm must accept single SNMP messages that have been recorded on a particular network location and it is supposed to assign these messages to sessions consisting of one or more SNMP messages.

2.4.1 Algorithm description

Before the steps taken are explained that are taken, it should be clear what information is stored during the process and in what form. Since multiple independent sessions can occur at the same time on a particular network, a list of so called *open sessions* must be kept. For each of these open sessions the following information is stored:

Open Session
• Operation type • Initiator party IP address • Initiator party port number • Other party IP address • Other party port number • {SNMP Message(s)}

Table 2.1: Stored open session information

The initiator is the source of the first message of a session. Besides this, there must also be a number of limits set on specific characteristics, like the maximum time between the original message and its retransmission. Following is a diagram showing the steps in the algorithm:

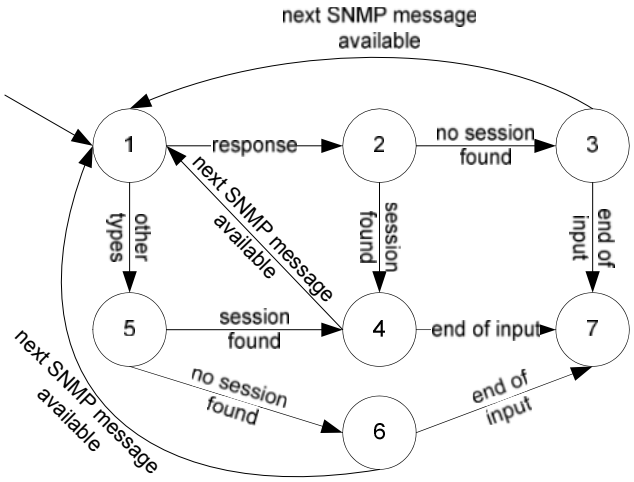

Figure 2.9: Steps in the session detection algorithm

The diagram shows the following steps being made:

Step 1
This is the start point of the algorithm. In this point, a single SNMP message is loaded from the input, assuming that the input has at least one SNMP message available in the beginning. Then, the type of the message is determined: is it a response message or it is something else?

Step 2
If the message is a response, then in the second point the open sessions are being tested to see which open session is 'willing' to accept this response message. This must be in accordance with the restrictions listed in the definition section for this kind of message. It should also be noted that this response may only be accepted when it is 'timely' (i.e. it must occur within a certain amount of time from the already listed respective request in that session). In the unlikely case that multiple open sessions are willing to accept this response message, shall the open session with the most recent addition of another message be considered the proper session.

Step 3
If no willing open session could be found, the response message is discarded and ought to be considered as unhandled. This could be the result of too restrictive timeout variables or it may be a response message of which the respective request has never been seen by the recording unit. This is especially likely at the beginning of a trace file. If other SNMP messages are available in the input to the algorithm, the next SNMP message is to be considered in point 1. If the end of the input has been reached, the next step will be point 7.

Step 4
In case a willing open session was found and selected for a message, then that message will be assigned to that one particular session only. If other SNMP messages are available in the input of this algorithm, the next SNMP message is to be considered in point 1. If the end of the input has been reached, the next step will be point 7.

Step 5
If a message was loaded in point 1 that was not a response message, this step will attempt to find a willing open session for this message. Hence, all open sessions are tested and only the case in which the open session will adhere to the restrictions stated in the definition section, shall that particular session be considered as the right one to which this message shall be added. It should be noted that for the operation types, for which the comparison of OIDs takes place, this comparison takes place between this message and the last response, if there is one available. The last response message is not per se the last response message chronologically, but is always the last response to the last request. In some cases, it may be possible that the

chronologically last response is a retransmission of an earlier response to a request that is not the last one in the session.

Step 6
If in point 5 no suitable open session could be found, then a new session will be created and added to the list of open sessions. If after this creation still more available SNMP messages exist in the input, then the algorithm will return to point 1, otherwise it will go to step 7.

Step 7
No more input messages are available. This means that no new SNMP messages can be added to any of the still open sessions. All of the open sessions will therefore be closed.

After the completion of step 7 shall the algorithm have yielded a number of sessions containing one or more messages. As explained before, in some exceptional case, it may be possible that a response message could not be linked to any of the open sessions. Only in those cases shall a message not be part of any of the resulting sessions.

2.4.2 Example algorithm execution

In order to give a better understanding of how all of these steps work together - yielding one or more sessions - follows in appendix A1 an example situation for this algorithm. That example is placed in a situation, which is not based on a real world trace, but is created artificially in order to highlight some of the most important aspects of this algorithm.

Chapter 3

Identifying session types

The previous chapter described the algorithm that would turn a raw set of SNMP messages into sessions. The next step in the process, which is described in this chapter, is the determination of session types and assigning this session type definition to a session. This will in turn form a basis for the next step: the detection of related sessions, forming session sets of a specific session set type.

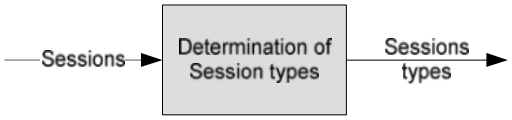

Figure 3.1: Input and output of this algorithm

Up to now, not much has been stated about the term session type. The only aspect that has been made clear is that it will be used in subsequent phases of the total set of algorithms. Still, in chapter 1.2 it has been noted that session types have the purpose of differentiating between 'equal' and 'different' sessions. In other words, it is supposed to be a method that supports other algorithms to signal whether or not two or more sessions are to be considered equal or different. If so called 'equal' sessions occur multiple times within the time frame of the recorded input, then clearly an algorithm is necessary that can determine whether two or more sessions may be considered equal.

But, when for example a manager requests the contents of a specific table of an agent every few minutes, not each of these sessions will appear to be exactly the same. There are a couple of reasons that cause this difference to occur. The most significant one will be that a table, or column thereof, may have changed in length during the course of a few minutes that makes up the time between two sessions. As a result a second table contents request may yield fewer, more or in some cases the exact same amount of request and response messages in both sessions. The session type is supposed to assist in this process of determining when two seemingly different sessions may still be considered equal and when not.

This chapter is built up from the same subsections as the previous chapter regarding the session determination algorithm. As a result, the following subsections will be covered in this chapter:

- A basic definition of a session type will be given, which is based on the discussed information of session types so far. This definition will not be complete;
- Then, some scenarios will be discussed that show the difficulty of determining session types for some cases. These case descriptions will be ended with suggestions for changes to the basic definition of a session type;
- Based on the basic definition of a session type and the case descriptions given in section 3.2, a complete definition of a session type shall be given in the third subsection. This will also form the basis on which the algorithm will be described in the next subsection;
- The last section describes the algorithm for identifying and assigning session types to any session.

3.1 Basic definition

The term session type was used in chapter 1.2 only briefly for introductory purposes. But, the usage of this term, as well as the depiction of session types in some diagrams, already gives some impression of the meaning of a session type. Based on this information, the following observations about a session type can be made:

- Sessions involving the same operation type and refer to the same set of columns, should be considered to be of the same session type;
- Sessions that are of the same session type may differ in length.

This results in the following basic definition of a session type:

Definition A *session type* is a type mark that can be determined for every session. Furthermore, sessions with the same session type:

- May be of variable length (i.e. may not have an equal number of member messages);
- Reference values from the same set of specific scalars or set of columns of tables.

This definition seems quite basic indeed. Hence, in the following subsection some scenarios shall be considered that will help to extend, or at least alter this basic definition into a complete definition covering the greater majority of possible sessions.

3.2 Considerations

Only the fact that sessions of the same session type may have a different number of messages has been considered so far. Following are a few important cases in which aspects of sessions and session types are highlighted that have to be taken into consideration before a complete definition of a session type can be given and even before the algorithm can be described.

In order to keep a uniform approach in these cases, a single scenario description will be given here. All cases that are discussed will use this scenario description as a basis.

Consider a scenario in which two managers exist alongside two agents. Consider also that the first agent has a large number of tables containing various values that can be referenced. The most important of these are the following single-column tables:

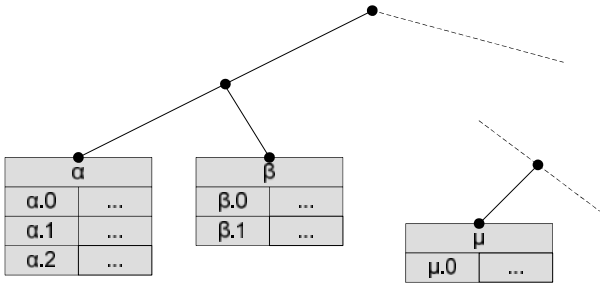

Figure 3.2: Highlighted tables of agent #1

This figure shows that at least an α-table exists containing three values. Besides that is a β-table that lexicographically follows the α-table and contains just two values. Finally, the μ-table, which is even past the β-table lexicographically speaking, contains just one item at all times.

Another agent also contains numerous referenceable tables and values. Still, the following tables are important in the following considerations:

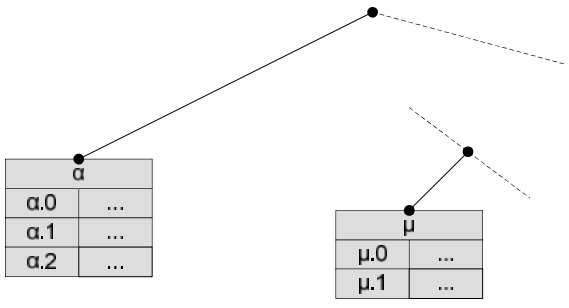

Figure 3.3: Highlighted tables of agent #2

This shows that the second agent also has an α-table with the exact same amount of listed values. It does not have a β-table like the first agent. Also, it has the μ-table, but in the case of this agent, this table may contain more than one value at a particular time.

The two managers are different in the sense that they operate from different IP addresses and may use different implementations of SNMP management software.

3.2.1 Different manager-agent relations

This first case will discuss three specific cases of comparable sessions involving different managers and agents that may not be considered the same. These are:

- Two managers requesting the contents of a specific table on a single agent;
- A manager requesting the contents of a particular table on the first agent and another manager requesting the contents of a table with the same OID as in the first manager-agent relation on a different agent;
- A single manager requesting the contents of one particular table on two different agents.

The following shows the messages making up two sessions that could occur between two different managers (*A* and *B*) and a specific agent (*C*):

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	2110	C	161	get-bulk	1	α
C	161	A	2110	response	1	α.0, α.1, α.2

Table 3.1: Session 1 occurring between manager #1 and agent #1

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
B	1440	C	161	get-bulk	1	α
C	161	B	1440	response	1	α.0, α.1, α.2

Table 3.2: Session 2 occurring between manager #2 and agent #1

Apparently, two independent managers have interest in the contents of the α-table of agent 1. These sessions look the same, but should they be considered as such: should they be marked with the same session type?

The same can be asked about sessions occurring between two different managers and two different agents that may have in common that both managers are interested in one specific table that is available at both agents. Take for example the following sessions that may take place between two managers (*A* and *B*) and two different agents (*C* and *D*).

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	1250	C	161	get-bulk	1	α
C	161	A	1250	response	1	α.0, α.1, α.2

Table 3.3: Session 1 occurring between manager #1 and agent #1

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
B	1880	D	161	get-bulk	1	α
D	161	B	1880	response	1	α.0, α.1, α.2

Table 3.4: Session 2 occurring between manager #2 and agent #2

Clearly, these sessions look the same, because at both agents the α-table contains the same number of items at that specific time. Also, the same operation type messages were used to retrieve the contents of this table. Still, should these sessions be considered of the same session type?

A final comparable case would be the involvement of a single manager requesting the contents of one particular table, the α-table, on two different agents. The following two sessions again suggest that they are of the same session type, but would this be a correct conclusion?

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	6420	C	161	get-bulk	1	α
C	161	A	6420	response	1	α.0, α.1, α.2

Table 3.5: Session 1 occurring between manager #1 and agent #1

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	5100	D	161	get-bulk	1	α
D	161	A	5100	response	1	α.0, α.1, α.2

Table 3.6: Session 2 occurring between manager #1 and agent #2

Although in all of these cases it would seem correct to state that a pair of sessions is of the same session type, however there are a couple of observations that would contradict such a basic comparison and conclusion of so called equality:

- The characteristics of tables with the same OID available on different agents may be different; i.e. the meaning or desired interpretation of the contents of a specific table may be different for different agents. Therefore, it remains questionable to state that two sessions - involving different agents with the same table references made - are comparable;
- On the other side, the cases involving more than one manager should incorporate the fact that the managers may be of different implementations and therefore they may have different implementation specific characteristics. This could result in problematic situations when, for example, periodicity is determined for specific sessions of a specific session type, since different managers may use different polling intervals and start times. As a result, apparently comparable sessions - involving different managers - should not be considered of the same type.

3.2.2 Different operations on the same table

Many sessions may reference the contents of only one particular table. But, these sessions may be of different operations. This subsection will discuss two sessions involving just this kind of cases and will continue addressing the question regarding when sessions should be considered of the same session type.

Consider a single manager *A* requesting the first scalar value of the μ-table on a single agent *C* through the use of a get request:

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	1092	C	161	get-req	1	μ.0
C	161	A	1092	response	1	μ.0

Table 3.7: Get session occurring between manager #1 and agent #1

Now, consider the same single manager requesting the change of the first value of the μ-table on a single agent through the use of a set request:

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	2109	C	161	set-req	1	μ.0
C	161	A	2109	response	1	μ.0

Table 3.8: Set session occurring between manager #1 and agent #1

These two different sessions take place between one manager and one agent. Also, they both involve the same OID that is being referenced. Based on the present definition of a session type, these two sessions would be regarded as of the same session type.

This example has shown that sessions involving completely different operation types could be considered of the same type. This is inappropriate because of the following:

- Since session types will be used to determine equality of sessions in the process of determining patterns in the occurrence of certain session types, it would seem incorrect to consider sessions involving two different operation types to be of the same session type. Also, because it would make no sense within the context of pattern detection of a certain session type, the set variant may, for example, occur significantly less often that the get variant does;
- Because equal session types means that the involved sessions have the same meaning, it would clearly be incorrect to state that, for example, a get session is equal to a set session simply because it involves the same set of column references.

3.2.3 Retransmissions within sessions

Already the possibility of sessions that are of the same session type having a different number of messages has been discussed. The reason that has been discussed so far would be that a table may change in length during the course of time, therefore two sessions involving the same table and that are of the same session type could have less, more or an equal number of messages. But, what about the case of retransmissions within sessions that involve the same table(s)? The following example describes such a situation.

Take again the previously described scenario, consisting of a single manager *A* and a single agent *C*. Following are two sessions that are the result of the manager querying twice the contents of the α-table. The sessions clearly ought to be considered part of the same session type, but the second session contains more messages as a result of retransmissions that do not occur in the first session.

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	4920	C	161	get-next	1	α
C	161	A	4920	response	1	α.0
A	4920	C	161	get-next	2	α.0
C	161	A	4920	response	2	α.1
A	4920	C	161	get-next	3	α.1
C	161	A	4920	response	3	α.2
A	4920	C	161	get-next	4	α.2
C	161	A	4920	response	4	β.0

Table 3.9: Get-next session occurring between manager #1 and agent #1 without retransmissions

Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
A	4920	C	161	get-next	1	α
C	161	A	4920	response	1	α.0
A	4920	C	161	get-next	2	α.0
A	4920	C	161	get-next	2	α.0
C	161	A	4920	response	2	α.1
A	4920	C	161	get-next	3	α.1
C	161	A	4920	response	3	α.2
A	4920	C	161	get-next	4	α.2
A	4920	C	161	get-next	4	α.2
C	161	A	4920	response	4	β.0

Table 3.10: Get-next session occurring between manager #1 and agent #1 with retransmissions

This example stresses again that sessions which should be considered of the same session type, may contain a different number of messages. The core observation of this example is:

- The occurrence of retransmissions should not affect the determination of the session type of a specific session. Clearly, the shown sessions should be considered of the same session type, even if one of them contains retransmissions.

3.3 Complete session type definition

After considering the scenarios listed in the previous subsection, it can be stated that the basic definition, as given in chapter 3.1 needs to be extended, in order to cover all the discussed cases and incorporate their respective points of interest. Taking the points from the basic definition, as well as the points that have been highlighted in every listed scenario, it yields the following complete definition of a session type:

Definition A *session type* is a type mark that can be determined for every session.
Furthermore, sessions with the same session type:

- Have a common set of OID prefixes;
- Have occurred between the same set of network elements, which are defined by their IP address and, if desirable, also their port number respectively;
- Are of the same operation type.

The first point have already been discussed in previous subsections and states that sessions of the same session type refer to the same set of scalars or columns of tables. Point two is a result of the fact that different agents may assign different meanings or expected interpretations to the values of table columns that may have the same OID. The pair of network elements between which sessions of a certain session type occur are identified through their IP addresses and in some cases also the port numbers of both sides. However, not in all cases, because in the case of many sessions, they are started by a manager which uses a different source port number for each session it starts. If that is the case, then the port number filter should not be applied for the manager side. In some cases, like a trap session, it may even be desirable to not filter on any of the port numbers, because in the case of a trap session it is the agent that initiates a session, which could choose a different port number for every trap session originating from that agent.

Point three is a result of the example shown in chapter 3.2.2. That example signified the need to differentiate between sessions involving different operations. As was stated earlier, a set session may be completely unrelated in its occurrence to the occurrence of a get session involving the same set of OID references. After all, this would give significant problems in future pattern detections, where these set and get operations would be considered the same in their occurrence.

The last example scenario of chapter 3.2 stressed the importance of disregarding retransmissions within sessions when a comparison is made on referenced OIDs between two or more sessions. This would leave the ‘normal’ set of request and response messages (if applicable) that make up the sessions which are to be scanned for OID reference comparison.

3.4 Algorithm

It is now possible to describe the developed algorithm, since the complete definition has been given and the most significant problem scenarios have been discussed. As has also been discussed in the introductory part of this subsection, this algorithm must accept sessions that are the result of the previous algorithm and it is supposed to return a session type for every possible session.

3.4.1 Algorithm description

The algorithm will, as the description already would suggest, handle one session at a time. So, sessions will be handled only sequentially. As a result, no information needs to be kept between the processing of two sessions. However, during the processing of a particular session, a list comprising of OID prefixes is kept. When, for example a get-next session requests the contents of all values in one particular column only, that list will at the end of the processing of that session only contain the respective OID of that column (the common OID prefix of the referenced column items). If on the other hand a get session is to be considered that would request only a specific scalar value of a column, then only that specific OID of that value would be listed. The purpose of that list is therefore to define which OID prefixes are referenced. A clear usage example of this list will be discussed in the next subsection which describes an example algorithm execution.

Besides the mentioned OID prefixes and the information about the operation type and the involved two network elements, there is no other data being used during the processing of a session. Following is a diagram that shows the steps taken in the algorithm in order to find the session type of each session given as input to this algorithm:

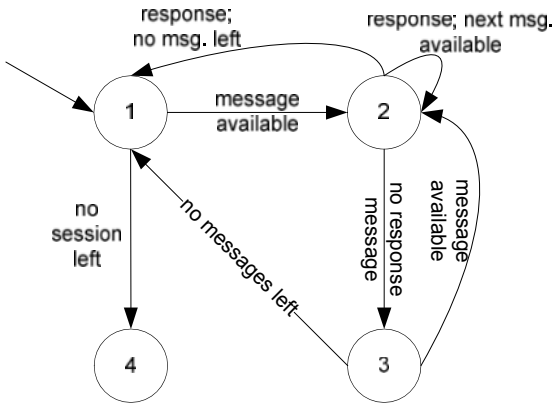

Figure 3.4: Steps taken in this algorithm

At the moment of starting the algorithm, the table of OID prefixes is still empty. Then, a set of sessions can be given as input to the algorithm for which for each of these sessions the respective session type needs to be determined. Following is a description of each of the steps taken, which in the end will yield a session type for each session:

Step 1

After starting the algorithm, the first session is loaded. This step will determine the involved network elements and the operation type of the loaded session. This information will be used later on. Also, if it is necessary, it will clear the table of OID prefixes that may still be left as a result of a previous processing of a session. Of course, at the very start, this table will be empty. Then, since every session always has at least one message, it will go to step 2. If no sessions are left in the input to this algorithm, it will go to point 4.

Step 2
This step will load the messages making up the session one at a time. It does so by beginning at the start of the list of messages of a session. If a loaded message is a response, it will load the next message until it either has reached the end of the list of messages of that session, or loads a non-response. In the latter case, the algorithm will go to point 3.

Step 3
This is a very important step. Here, the non-response message is loaded and its OIDs are being compared with those of the last response of that session that has been loaded so far. The following sub steps are taken here:

- First, the last response message is searched for that is part of the list of messages of this session that have been loaded so far. The last response is found using the same method as in the previous algorithm. This last response is thus not just the chronologically last response, but the last response to the last non-response message;
- Then, the OIDs between this non-response message and the found response message are being compared for equality. All OIDs that are found in the latest non-response message and not in the found response message will be taken to the next step. Note here that if no last response could be found, all OIDs of this non-response message will be taken to the next step;
- The remaining OIDs are either explicit references to a column item, or just a column. These OIDs are considered 'new' and are therefore considered as a new reference. This sub step will test to see if any of these remaining OIDs is a reference to a column or scalar of which already a reference is listed in the list of found OID prefixes. If that is the case, than that particular OID will not be added to the list of found OID prefixes. This check makes use of a prefix comparison, which will be shown in an example later on;
- Any remaining OIDs, for which there are no OID references which are already listed in the list of found OID prefixes, will at this point be appended to this list.

After finishing these sub steps, the algorithm will return to step 2, where it will continue loading messages of this session. If however no other messages are available, the algorithm will return to step 1;

Step 4
This point marks the end of the algorithm, where it will result all the session types for every respective session that was given as input to this algorithm.

When a table retrieval session, using for example get-next messages, is being handled by this algorithm, it will result in the OID prefixes of the columns or scalars that have been referenced. This, because the first request of the session will reference the respective columns for the first time and all subsequent requests are based on the returned OIDs of the last response occurring before that. Some cases for OID prefix determination will be discussed here:

- In the case of an operation other than get-next and get-bulk, the referenced OIDs in the initial message are not likely references to column(s), but rather direct references to column values / scalars.
    - In the case of a trap, inform or possibly a report session, this is obvious, since it is the initiating party of that session that has full knowledge of the tables, columns and their values. Hence, the OIDs referenced in these cases are direct references to column items / scalars;
    - Set sessions should also be considered as such, because either the manager is programmed to reference specific column items / scalars, or it has retrieved that knowledge through other sessions, like a get session. This is determined, based on experiments with trace data from various locations.
- In the case of get-next and get-bulk, the OIDs are generally references to the columns themselves. Even if specific values were to be retrieved through either of these operations, it is very likely that the manager has been explicitly programmed to retrieve those specific values and will therefore refer to these same instances in the future, if it is programmed to do so. As a result, in these cases, the prefix of the OIDs in those non-response messages will be taken before they are being compared with any of the listed OIDs in the list of OID prefixes. This will avoid specific references to values / scalars being listed, which may change during the course of time, while the

meaning of factually equal sessions may be the same. Consequently, no unnecessary new session types will be defined for these sessions that are actually the same.

In general, it should be added that in any case an OID is found that is a parent of any of the listed OIDs in the list of OID prefixes during a point in time of the algorithm execution, then it will replace the child OID which is listed in the list of OID prefixes. If after that still more OIDs in the list of OID prefixes are found that are children of this new parent OID, then they will be removed.

After the algorithm has completed processing a particular session, it will yield a session type for that session. This session type is made up of the following data fields, which make it possible to discriminate between different sessions and allow for easy comparison for equality of sessions in the next algorithm.

Session Type
Operation Type
Initiator IP Address
Initiator Port Number *
Other Party IP Address
Other Party Port Number *
{OID prefix(es)}

Table 3.11: Information used to identify a session type

Note that two fields contain asterisks. This is because these fields may in some cases not be defined as a result of the operation type of the session type, which has been discussed earlier.

In the subsequent parts of this thesis, only a reference may be made to a specific session type. This will always be of the following format:

OPERATION_TYPE-INITIATOR_IP[INITIATOR_PORT]-OTHER_IP[OTHER_PORT]-[INDEX]

- OPERATION_TYPE: the operation type of the session type;
- INITIATOR_IP: the IP address of the initiator of the sessions that have this session type;
- INITIATOR_PORT: the port number of the initiator. This field may be disregarded, if desired;
- OTHER_IP: the IP address of the other involved party of the sessions that have this session type;
- OTHER_PORT: the port number of the other party. This field may be disregarded, if desired;
- INDEX: for the same set, consisting of an initiator and another party and a specific operation type, there may be multiple session types. Hence this integer field will allow for differentiation between session types of this sort.

3.4.2 Example algorithm execution

In order to give a better understanding of how all of these steps work together - yielding the respective session types for each session given as input - follows in appendix A2 an example situation for this algorithm.

Chapter 4

Finding session sets of the same session set type

At this point the definitions and the respective algorithms for sessions and session types have been discussed. It is the purpose of the next step to find relations between the occurrences of certain sessions. As has been stated previously, this requires the ability to determine when a session is to be considered 'equal' to another session. This has been achieved through the use of session types. Hence, this next algorithm will accept sessions and their respective session types and will attempt to find relations in the occurrences of these sessions.

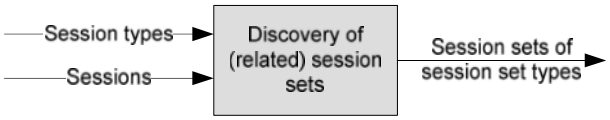

Figure 4.1: Input and output of this algorithm

The attempt to find relations between the occurrences of sessions is a result of a previous observation made in chapter 1.2, where it was mentioned that sessions are likely related to one or more other sessions. This is the result of the fact that, for example, a manager is set to poll a list of agents every few minutes. Such polling behaviour results in the same set of sessions, which occur every few minutes. Also, since a manager may poll multiple agents, multiple sessions of a certain type may always occur within a certain amount of time from each other. Once the relationship between one or more session types has been identified, this shall result in a session set type describing session sets that are comparable and related. These two definitions shall be discussed and further introduced in the next subsection.

This chapter gives an in-dept description of the involved definitions and the exact operation of this algorithm. The following aspects shall be discussed in this chapter:

- First, an introductory example shall be given to explain the purpose and general expectations of this algorithm;
- Then, a number of real world considerations shall be discussed, that will give a better understanding of the exact desired result of this algorithm;
- Based on the definition of the involved terms, as well as the given real world consideration, the algorithm shall be discussed.

4.1 Introductory example

Although the terms *session set* and *session set type* have already been illustrated in chapter 1.2 and mentioned in subsequent chapters, a simple example will now be given that will only show the expected purpose and results of this algorithm. A complete definition of these two shall be given later on in this chapter.

Consider an example scenario with the following characteristics:

- Manager *A* polls two agents (*B* and *C*) every 300 seconds;
- This manager is only interested in the values listed in the columns with OID α or β;
- It requests these values through the use of sequential get-next request messages.

Following is an overview of the sessions that may take place during a period of about 600 seconds.

Start Time (sec.)	Session type
1,2	get-next-request-A[]-B[161]-[0]
1,8	get-next-request-A[]-C[161]-[0]
301,1	get-next-request-A[]-B[161]-[0]
302,0	get-next-request-A[]-C[161]-[0]
601,3	get-next-request-A[]-B[161]-[0]
603,1	get-next-request-A[]-C[161]-[0]

Table 4.1: Two session types occurring both three times

This simple example overview shows the occurrence of six sessions that have been identified with the first algorithm. The second algorithm has identified the session types of each of these sessions. Now these sessions and their respective session types are given as input to this algorithm.

Since this algorithm is expected to find relationships between the occurrences of certain sessions, it should see in this very simple case that the two session types always occur within seconds from each other. Furthermore, they are all started by the same initiator *A*. So, by observing just these six sessions, it should be the result of this algorithm to state that all occurrences of the two session types are related. In that case, the following session sets can be identified:

Start Time (sec.)	Session type
1,2	get-next-request-A[]-B[161]-[0]
1,8	get-next-request-A[]-C[161]-[0]

Table 4.2: Session set #1

Start Time (sec.)	Session type
301,1	get-next-request-A[]-B[161]-[0]
302,0	get-next-request-A[]-C[161]-[0]

Table 4.3: Session set #2

Start Time (sec.)	Session type
601,3	get-next-request-A[]-B[161]-[0]
603,1	get-next-request-A[]-C[161]-[0]

Table 4.4: Session set #3

Since all of these session sets have the exact same session types listed, these three session sets will be considered to be of the same session set type:

Session set type #1
Session set #1
Session set #2
Session set #3

Table 4.5: Session set type #1

In this case, the related session types occur within a very short time span of each other. But, there may be cases where this is not so obvious. As a result, it should be stated that all occurrences of session types within a session set occur within a specific range of time of each other. This means that there can be no gaps larger than a certain amount of time between the sessions within a single session set; i.e. all sessions making up a particular session set should follow each other within a certain amount of time. If not, than there may be reason to believe that a new session set is started. After all, when a manager, as in this case, performs regular polling behaviour, then all sessions in a single polling instance follow each other very

fast. For example, after a manager completes the retrieval of a specific table, it will either immediately go on with the next table on an agent, poll the next agent in its list, or stop and wait for the next point in time where it is supposed to poll its list of agents.

This simple example has shown some important characteristics of the two definitions. The observations that have been made so far about both of these definitions can be summarized as follows:

A *session set* encompasses:

- One or more sessions that occur within a certain time frame of each other;
- These sessions are to some extend related to each other.

A *session set type* encompasses:

- One or more session sets that have a certain level of equality with respect to the occurrence of session types in these respective session sets.

The general meaning of the two involved definitions for this algorithm have now been discussed. Also, it has become clear that this algorithm has the task of finding session set types involving related session sets. These session set types and respective session sets will be used as a basis for the next and final algorithm.

The given example is very simple and is by no means complex enough to deal with real world problems that may be encountered in traces. The following subsection shows some problem scenarios that have to be addressed, before an exact definition of the two mentioned terms or the algorithm description can be given.

4.2 Considerations

Up to now only a very simple example has been shown, which clarifies the general purpose of this algorithm and the steps that have to be taken within this algorithm. But, as has been said, this is just a very simple case, which only consists of a single manager which generates obviously periodic session sets of a single session set type. It is the purpose of this subsection to show a number of different scenarios and aspects that a real world trace may have, which could yield some difficulties in the determination of some session sets and session set types. Every scenario shall be ended with the aspects that have to be taken into account for the complete definition of a session set type and its member session sets.

In order to keep a uniform approach in these cases, a general scenario description will be given here. All cases that are subsequently discussed, will use this general scenario description as a basis.

Consider a scenario in which two managers exist alongside a single agent with the following specifics:

- The first manager *A* is programmed to retrieve all values from column α on a periodic basis, based on a specific preset interval;
- The second manager *B* is programmed to retrieve all values from both column α and β on a periodic basis, based on a specific preset interval;
- Agent *C* has a large number of tables containing various values which can be referenced. The most important of these are the following tables, which are the same as in the previous algorithm description.

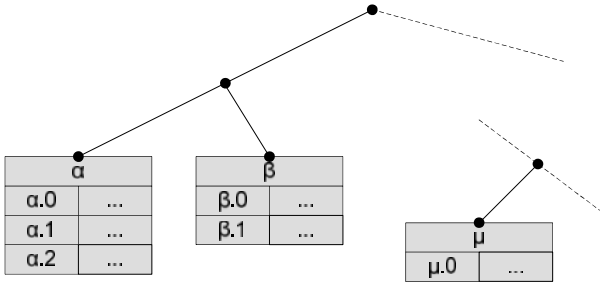

Figure 4.2: Highlighted tables of agent #1

4.2.1 Multiple initiators

The already stated example in the introductory section involved only a single initiator party for every session. This example scenario shows the potential difficulties that arise when in a trace file more than just one initiator exists.

Consider the case in which both mentioned managers are actively polling the specified agent. The following session types could occur within a specific time span:

Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]
0,2	get-next-request-B[]-C[161]-[0]
0,5	get-next-request-B[]-C[161]-[1]
150,1	get-next-request-A[]-C[161]-[0]
300,0	get-next-request-B[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]
300,7	get-next-request-B[]-C[161]-[1]

Table 4.6: Occurrence of session types with multiple initiators

This table shows a segment of a potential trace file, translated to the session types of each occurring session. Although this is a recording of just a little more than 300 seconds, the problem that arises with the involvement of multiple initiators becomes clear: which session types should be considered related?

A first observation can be made regarding the fact that initiator and manager *B* uses a different interval to poll its desired columns on the agent, than the other manager *A* does. As a result, the sessions with session types that are initiated by different parties should not be considered related. If this would not be stated, then it could be possible to conclude that session types *get-next-request-A[]-C[161]-[0]*, *get-next-request-B[]-C[161]-[0]* and *get-next-request-B[]-C[161]-[1]* are related, because in two cases all of these session types occur within a reasonable time of each other. But, this would not cover the occurrence of session type *get-next-request-A[]-C[161]-[0]* at time point 150,1 seconds.

Another possibility of session type relations would then be that the sessions that were initiated by different initiators are considered separately. This seems correct, since a manager is programmed to perform a certain behaviour, independent of the programmed behaviour of a possible other manager; i.e. the polling behaviour of manager *A* is strictly independent of that of manager *B*. A manual observation of a few trace files suggests this to be a correct assessment.

When the session types are first filtered and separated by initiator IP address, then it will be obvious to state that the following two sets of session types contain session types that are related in their occurrence:

get-next-request-A[]-C[161]-[0]

get-next-request-B[]-C[161]-[0]
get-next-request-B[]-C[161]-[1]

The next step, as has been stated in the introductory example, is the identification of session sets. Since these contain occurrences of session types that are related to each other, it is possible to define the following session sets:

Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]

Table 4.7: Session set #1

Start Time (sec.)	Session type
0,2	get-next-request-B[]-C[161]-[0]
0,5	get-next-request-B[]-C[161]-[1]

Table 4.8: Session set #2

Start Time (sec.)	Session type
150,1	get-next-request-A[]-C[161]-[0]

Table 4.9: Session set #3

Start Time (sec.)	Session type
300,0	get-next-request-B[]-C[161]-[0]
300,7	get-next-request-B[]-C[161]-[1]

Table 4.10: Session set #4

Start Time (sec.)	Session type
300,1	get-next-request-A[]-C[161]-[0]

Table 4.11: Session set #5

Now, it is possible to define the following session set types:

Session set type #1	Session set type #2
Session set #1	Session set #2
Session set #3	Session set #4
Session set #5	

Table 4.12: Session set types #1 and #2

This scenario has shown the following topics that should be taken into account for the exact definition of the terms session set and session set type and also the algorithm itself:

- Before attempting to find which session types are related in their occurrence, a filter should be applied, which separates different sessions, initiated by different initiators.

4.2.2 Composite of periodic and aperiodic behaviour

This example will show some of the most profound difficulties regarding a situation in which both periodic and aperiodic traffic intertwine with each other.

Take for example the extended scenario of the one previously considered and add the case that besides the regular polling of both managers, also manager *A* performs an aperiodic get-request of the first scalar value of the μ-column. This may result in the following overview of detected sessions and their respective session types:

Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]
0,2	get-next-request-B[]-C[161]-[0]
0,5	get-next-request-B[]-C[161]-[1]
71,0	get-request-A[]-C[161]-[0]
150,1	get-next-request-A[]-C[161]-[0]
155,8	get-request-A[]-C[161]-[0]
298,6	get-request-A[]-C[161]-[0]
300,0	get-next-request-B[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]
300,7	get-next-request-B[]-C[161]-[1]
328,5	get-request-A[]-C[161]-[0]

Table 4.13: Occurrence of session types with aperiodic addition

This table shows the same set of detected occurrences of session types as in the previous scenario, with the addition of four sessions that are of a session type that seems to perform a behaviour that cannot be matched with any of the already known session set types and their respective session sets that were found earlier.

What becomes clear from this listing is that just filtering the initiator IP address will not solve the problem for this scenario, as it did in the previous one. This is a result of the fact that even after such filtering, the newly added session type still seems to be related to the occurrence of some session sets, because it occurs within a reasonable amount of time within those existing session sets. But, on the other hand, this newly added session type occurs more often and there are two cases where its occurrence is clearly unrelated to any other session type, namely the occurrence at 71,0 seconds and at 328,5 seconds. In these two cases, there are no other session types that seem to relate to any of these two occurrences; they seem to appear independent. This could, for instance, be the result of a manager being ordered to retrieve a certain value by the user of the manager application.

A result of these observations is that the newly added session type should be considered as independent in its occurrence, based on the information given here. This would leave the already mentioned session set types and their respective session sets unchanged and would add a new session set type consisting of four session sets that all contain only one instance of this session type.

This scenario, in which besides regular polling traffic, also clearly aperiodic traffic occurs from the same IP address as the one that generated periodic polling traffic, has highlighted the following aspect:

- When a relationship is looked for, the proximity in occurrence of session types should not only be considered, but also the number of occurrences of each of the involved session types. This will avoid session types being considered part of a session set, where they actually are not;
- The given example of this subsection also shows that the order of occurrence of session types may not always be the same for every session set of a particular session set type. This may be the result of parallel polling by a manager. In such a case, certain sessions may be seen before another session from the perspective of the recording unit.

4.2.3 Multiple managers operating from the same IP address

Another significant difficulty is the possibility that two or more managers may operate from the same IP address from the perspective of the recording unit (that creates the trace files) which is part of a network. This problem shall be discussed here by using the example scenario given in the introductory part of this subsection.

Before an example is given, it may be of interest to show the two most likely cases in which multiple managers may operate from the same IP address:

- Two or more managers operate from a different IP address behind a NAT, in which case the recording unit is placed on the other end of the NAT, which only sees a single IP address;
- Two or more managers operate as different applications on a single system with a single IP address.

In either of the two cases, it is not easily possible to determine whether there are indeed multiple managers operating from a single IP address, or that it is just one manager with potential odd settings. This is so difficult, because most manager applications tend to use different port numbers on their side for every new session. Also, in the case multiple managers operating from the same IP address, the only obvious distinction will be their port number. However, by simply looking at the sessions initiated from a specific IP address, it is not easy to state which session is started by which manager. This can have consequences for this algorithm, if there are managers that reference the same columns on a specific agent by using the same operation type, or worse, they have an overlap in session types and also use a specific interval for their polling. This may result in multiple sessions of the same session type to occur simultaneously. In order for this problem to occur, it is necessary that all involved managers operating from a single IP address use a different port number for every session. Otherwise, it could still be possible to filter the sessions based on the initiator's port number.

Consider the following listing of recorded sessions:

Start Time (sec.)	**Session type**
0,1	get-next-request-A[]-C[161]-[0]
0,2	get-next-request-A[]-C[161]-[0]
0,5	get-next-request-A[]-C[161]-[1]
150,1	get-next-request-A[]-C[161]-[0]
300,0	get-next-request-A[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]
300,7	get-next-request-A[]-C[161]-[1]

Table 4.14: Occurrence of session types with aperiodic addition

This table shows that where previously three different session types were detected, there are now just two. As a result of the fact that the two managers now operate from a single IP address and use a different port number for every session, it results in the apparent double occurrence of session type *get-next-request-A[]-C[161]-[0]* around 0 seconds in the trace and around 300 seconds. It is now no longer possible to distinguish which manager started which session.

A possible solution to this very difficult scenario could be to not only look at a single occurrence of a session type, but more on a higher level approach, in which one should look at all of the occurrences of session types and whether then a specific kind of relation can be made, creating session sets that are comparable and of the same session set type.

In this case, though nothing is known about the session types at later times of this trace, it would still be possible to state that two instances of session type *get-next-request-A[]-C[161]-[0]* and a single instance of the session type *get-next-request-A[]-C[161]-[1]* seem to be occurring twice in this small trace segment. Besides that, a single occurrence of the session type *get-next-request-A[]-C[161]-[1]* can be found around time mark 150,1 seconds. Thus, by not just looking at a single occurrence of a session type within a potential session set, in combination with the more general look at the trace itself, yields the following sessions sets that make up two different session set types:

Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]
0,2	get-next-request-A[]-C[161]-[0]
0,5	get-next-request-A[]-C[161]-[1]

Table 4.15: Session set #1

Start Time (sec.)	Session type
150,1	get-next-request-A[]-C[161]-[0]

Table 4.16: Session set #2

Start Time (sec.)	Session type
300,0	get-next-request-A[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]
300,7	get-next-request-A[]-C[161]-[1]

Table 4.17: Session set #3

Session set type #1	Session set type #2
Session set #1	Session set #2
Session set #3	

Table 4.18: Session set types #1 and #2

Besides the mentioned difference in trying to find relations, the general method of only considering instances of session types that occurred within a specific time frame of each other, was also taken into account in the above listed session sets. With this is meant that the session starting at 150,1 is not considered a reasonable continuation of the first session set, assuming that the time between the end of the first session set and the mentioned session is large enough, in order to conclude that this is indeed a new session set.

Note that this problem would also arise if two different agents operate from the same IP address. Manager initiated sessions are hard to differentiate between as has been discussed earlier, but it will be much more difficult in the case of an agent initiated session, since the source port of that session will be indistinguishable from other agent initiated sessions operating from the same IP address. However, manual inspection of some trace files suggests that still a large number of agent initiated sessions use the standard agent port number (usually 161 [6]) as the initiator port number.

The following observations can be taken as a result of this scenario description:

- Session sets of a specific session set type may contain multiple occurrences of the same session type;
- Session sets making up session set types may contain session types which do not all occur an equal amount within the respective session sets, nor in a trace.

4.2.4 Irregularly occurring session types

Another very difficult scenario is one that contains irregularly occurring session types within apparent session sets that are of a session set type. This may be the result of, for example, a manager polling an agent on a regular basis, but not every polling instance results in the exact same set of session type occurrences. This phenomenon raises the question: how to find relationships between occurrences of session types, when not every session type may occur in every session set of a session type?

To give a better understanding of the problem at hand, consider the following list of detected occurrences of session types in a fictional trace:

Start Time (sec.)	Session type	Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]	600,2	get-next-request-B[]-C[161]-[0]
0,2	get-next-request-B[]-C[161]-[0]	601,2	get-next-request-A[]-C[161]-[1]
0,5	get-next-request-A[]-C[161]-[1]	899,8	get-next-request-B[]-C[161]-[0]
300,0	get-next-request-B[]-C[161]-[0]	900,1	get-next-request-A[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]	900,4	get-next-request-A[]-C[161]-[1]
300,7	get-next-request-A[]-C[161]-[1]		

Table 4.19: Occurrences of session types with some irregular occurrence

This example trace would suggest that the first three, the second three, the following two and the last three session type occurrences all yield different session sets each.

Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]
0,2	get-next-request-B[]-C[161]-[0]
0,5	get-next-request-A[]-C[161]-[1]

Table 4.20: Session set #1

Start Time (sec.)	Session type
300,0	get-next-request-B[]-C[161]-[0]
300,1	get-next-request-A[]-C[161]-[0]
300,7	get-next-request-A[]-C[161]-[1]

Table 4.21: Session set #2

Start Time (sec.)	Session type
600,2	get-next-request-B[]-C[161]-[0]
601,2	get-next-request-A[]-C[161]-[1]

Table 4.22: Session set #3

Start Time (sec.)	Session type
899,8	get-next-request-B[]-C[161]-[0]
900,1	get-next-request-A[]-C[161]-[0]
900,4	get-next-request-A[]-C[161]-[1]

Table 4.23: Session set #3

But, a closer inspection of the session types making up these session sets shows that a difference can be detected. Of the four session sets there are three logically of a single session set type, namely: the first, second and fourth. The third session set misses an occurrence of session type *get-next-request-A[]-C[161]-[0]*.

Irregularly occurring session types and therefore unequal session sets of the same session set type can be the result of implementation specific aspects of, for example, the manager application. There may be scenarios in which the manager decides to also request the values of another column or table, as a result of the fact that a specific column item is of a value that requires further inspection.

A very simple example cause of such a scenario would be the following:

- A single agent is a network printer containing referenceable tables;
- A single manager set to poll certain tables on this agent every few minutes;

- If a certain table value of the agent suggests that there is a printer problem, then this manager will also request the contents of a different table related to, for example, the amount of paper left in each input tray, or the amount of toner left in each cartridge.

Hence, in the case a printer error is detected by this manager, it will generate more sessions, likely of a different session type than usually occur in such a polling instance. This will result in irregularly occurring session types within a polling instance.

Such a scenario, like the one described above, suggests that in some cases it would be desirable to consider the irregularly occurring session type also as a member of some of the session sets that make up a session set type. But this would raise a new question: in how many session sets does a session type need to occur in order for it to be considered part of a session set of a specific session set type? When this number of required occurrences is taken too low, it may result in session sets that contain the occurrences of session types that are in fact not related in their occurrences to the other listed session types for that session set. On the other hand, when this number is taken too high, it may incorrectly consider related (almost) regularly occurring session type occurrences not to be part of a session set.

So, in the case of the example trace listing given in this subsection, it may be concluded that if a session of session type *get-next-request-A[]-C[161]-[0]* occurs often enough in session sets, then all four listed session sets may be considered of the same session set type. If a more restrictive approach would be taken, then this example trace description would result in two different session set types, one involving the first, second and fourth session sets and a second session set type that only involves the third session set.

The following can be stated, based on the observations made in this subsection:

- Certain session types may not always occur in every session set of a specific session set type;
- The respective sessions of irregularly occurring session types may, depending on the trace characteristics, be considered part of a session set;
- This measure may result in session sets of a single session set type that contain a variable number of session type occurrences.

4.2.5 Incomplete session sets

The final consideration which is being discussed is the occurrence of incomplete session sets. Where the previous subsection discussed the irregular addition of session type occurrences to the session sets, this section will cover the cases in which fewer sessions in session sets can be found.

There are a few causes for incomplete session sets. One of them is related to the starting and the stopping of trace recordings. Since every trace recording may start and/or stop at a time during which a session is active, this may result in an incomplete recording. Therefore, the very first and the very last session of a trace recording may be incomplete. Take for example the case in which a single manager of a network operates alongside a single agent. The manager is interested in the values of the α-column on a periodic basis. Then, the following recording may take place on such a network:

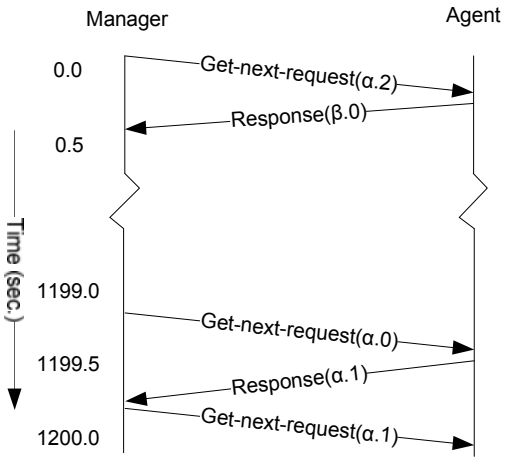

Figure 4.3: TSD highlighting border traffic issues

This diagram shows that the beginning and the end session are likely incomplete. As a result, the very first and very last session may be considered of a session type that may not be found in any other position in the trace. This may also affect the determination of a session set at the beginning and the end of a trace. Therefore, not only should the 'border' session be discarded before they are taken into consideration in this algorithm, also the very first and very last session set may need to be discarded, in order to avoid session set membership to an incorrect session set type.

Another cause of incomplete session sets could be that a regular polling instance, initiated by a certain manager, takes so long that the time reaches the point where the manager is programmed to start its next polling instance. The following graph shows this possibility:

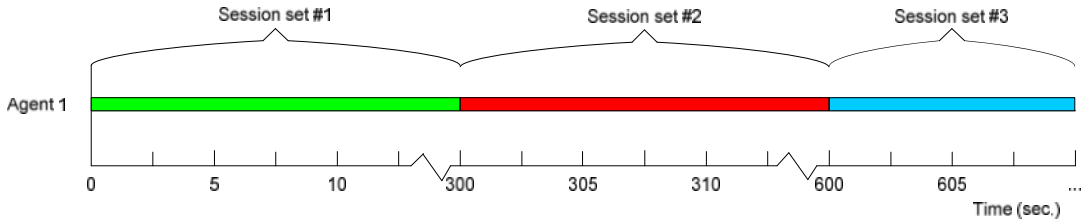

Figure 4.4: Graph showing the possibility of incomplete session sets

Either the time between the polling instances is taken very short, or the timeout limits are excessively large. In these cases, a manager could choose to abort a running polling instance and start the new one. This has two consequences: the first mentioned polling instance did not complete, which may result in a too small session set later on, and the time between two successive session sets may be very short indeed, which will make it hard to distinguish between two session sets.

The following can be concluded, based on these two possibilities of incomplete session sets:

- The very first and the very last session and session set may need to be discarded, in order to avoid possible matching problems with session set types;
- The time between session sets that may make up the same session set type may be very short.

4.3 Definitions

In the introductory chapter the two terms session set and session set type have already been discussed briefly. Following are the two definitions and the respective characteristics of these two terms that have been based on the scenarios discussed in the previous subsection.

Definition A *session set* encompasses one or more sessions that have the following characteristics:

- All occurred within a initiator specific time frame of each other;
- Are related in their occurrence to each other;
- Are all initiated by a single initiator.

Definition A *session set type* involves one or more session sets which:

- Are all initiated by a single initiator;
- All have a very high (initiator specific) percentage of session type occurrence equality (i.e. a particular session type shall occur in (almost) all member session sets).

4.4 Algorithm

The given definitions of the two terms involved in this algorithm show that it is very difficult to find an algorithm that is able to cope with all of these scenarios. Nevertheless, an algorithm has been developed, which will be discussed thoroughly in this subsection. Moreover, the description of this algorithm shall be followed by an extensive example which will incorporate many of the already stated difficulties, in order to give an even better understanding of this algorithm.

4.4.1 Algorithm description

The algorithm that has been developed to find session set types involving of session sets shall be discussed here. The steps taken in this algorithm will be described alongside an explanatory abstract example.

Before the algorithm can be described or even considered, it is necessary to define a number of variables as a result of the observations made in the previous subsections of this chapter. The first one is a timeout value that has to specify the maximum amount of time between the end and start of any two sessions that are supposedly part of a session set. If the time between two sessions is larger than this value, it would not be reasonable to consider those sessions to be related. On the other hand, this value should also not be taken too small, in which case a new session set may be considered, where it is actually a continuation of an existing session set. As a result, in normal traces, this is the minimum time between session sets that should make up a session set type. Take for instance the polling behaviour of a manager. This manager may initiate a session set every 300 seconds which each take 10 seconds to complete. Then, the timeout value should be taken smaller than 290 seconds, in order to consider the session sets correctly. Note that this value may be different for different initiators of sessions within a trace. But, for the sake of clarity, the timeout value for the example shall be taken constant for all initiators at 3 seconds.

Another value that should be stated beforehand is the minimum amount of occurrences of a session type before it will be considered part of the respective session sets that make up a larger set of session sets of the same session set type. Also this value may be initiator specific or trace specific, but in this case it will be set to 80%. Its purpose will be made clear later on.

Also, the maximum difference in length of session sets that belong to a single session set type should be determined at this point. For simplicity, this will be set to ± 50% of the average length of the session sets making up a single session set type. The exact calculation and involved variables of this value shall be discussed later on.

Now, consider the case that a single manager is polling exactly 12 agents every 300 seconds. All of the involved traffic has been recorded for a specific duration. Also, to show some of the characteristics of this algorithm, an additional session is added at time marker 24, which is generated manually by the user of the management application. Following is an overview of the first 620 seconds of the trace, translated to blocks which represent individual sessions:

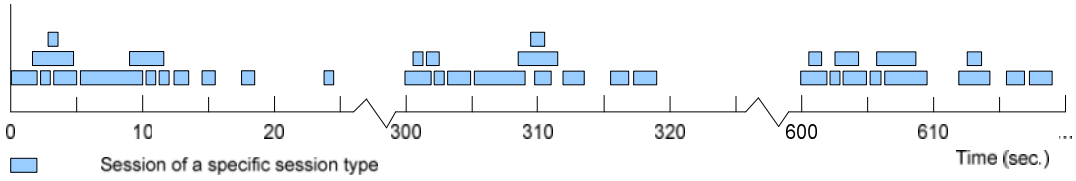

Figure 4.5: Sessions occurring within the first 620 seconds of an artificial trace

Note that some space can be found between sessions, which would indicate possible processing time at the manager side. Also, note that some sessions occur simultaneously. This abstract example shall now be used to explain the algorithm.

The following flow diagram describes the many steps of the algorithm from a more abstract view. The meaning of every step and condition shall be made clear during the course of the next few pages:

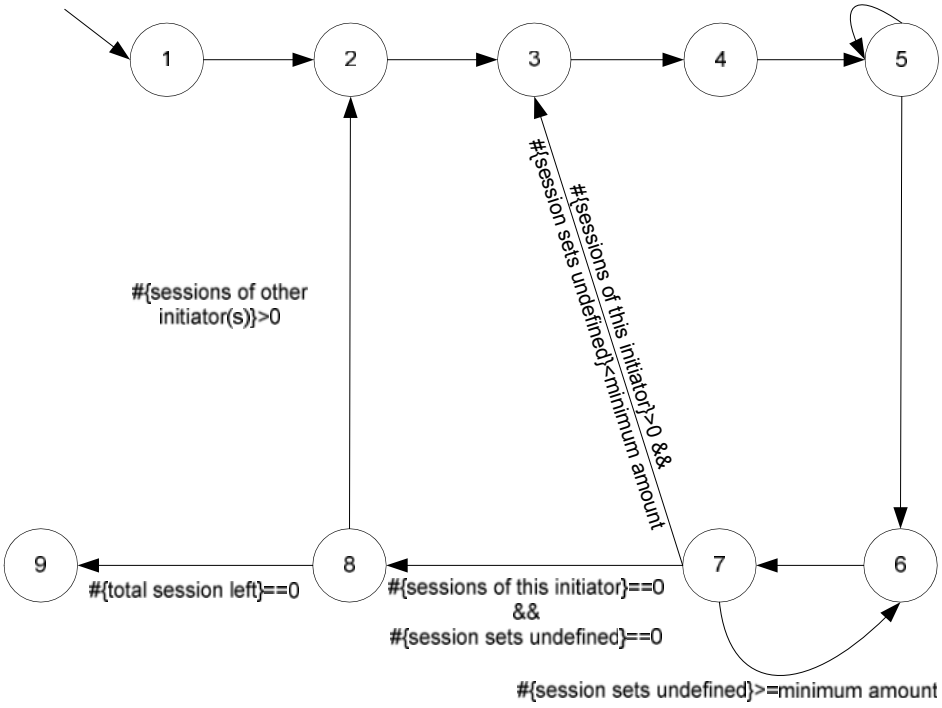

Figure 4.6: Steps as part of this algorithm

1. Remove incomplete 'border' sessions

As has been discussed in the previous subsection, there is a need to remove the incomplete border sessions as a result of starting and stopping the recording of traffic on a network.

One of the measures includes that a specific amount of time from either border (start and end marker of a trace) needs to be chosen, in order to cope with the time between retransmissions of messages. If, for example, a manager waits 20 seconds before it sends a retransmission and just after a manager has sent its message the recording starts. Then, it may be possible that the retransmission is seen as the first message of a 'new' session almost 20 seconds after the start of the trace recording.

Once these sessions have been selected, it may very well be possible that there is nothing wrong with these recorded sessions, because they may in some cases be complete. In those cases, these sessions will be of a certain session type that will occur at least twice in the whole trace. If they are of a session type that only occurs a single time in a trace, it is most likely that something of that session was not recorded. Hence, in the latter case, the session should be removed before any further algorithm steps are taken.

These two filters can be summarized as follows: in order to remove incomplete border sessions, one should remove all sessions that are within a certain amount of time from either border and are of a session type that only has one occurrence in the whole trace.

2. Filter per initiator IP address

The second step, after the sessions and their session types have been given as input to this algorithm, is to select one of the available initiator IP addresses of one of the sessions in the trace and filter all sessions, such that only those sessions remain that have been initiated from a single IP address. The abstract overview given in figure 4.5 shall remain unchanged after this filter is applied, since in that example only one initiator IP address exists: the single manager.

3. Count session type occurrences

The next step is to count all occurrences of each remaining session type. Generally, in the case of a manager that is set to poll a few agents on a regular basis, this would yield a list of session type occurrences that are very close to each other in numbers. The main difference would be the result of the start and stopping of the traffic recording, which could result in a missing of a certain session type occurrence in either the very first and/or the very last session set. In the worst case, the difference between any two session types would in such a case be two (the missing of a session type instance in the very first session set and also in the very last session set).

4. Find the largest group of session types that seem directly related

This step is more difficult than the previous three. This step will use the generated overview of the number of occurrences of each session type as input. It will try to find the largest possible group of session types that, within the trace, always occur within the stated timeout of each other and all occur equally often in trace and all occur the least number of times. This will result in a list of session types that are considered to be directly related.

The first question to be answered in this approach is about the reason for looking for these session types. As has been stated in the previous subsection, there must remain room for the possibility that a number of session types do not always occur in every session set of a possible session set type. Still, there must be a group of session types that do always occur in every session set of a session set type. This method aims at finding those so called *core session types*, because they can be found in every session set that makes up a session set type.

The second likely question that needs to be answered is about why the session types are picked that have the least number of occurrences, instead of the session types that occur the most often. Although this latter case would seem more plausible in scenarios in which there would be certainty that no two managers are operating from the same IP address. A manager could, for example, be set to have two sets of polling settings that are used independently of each other where there is an overlap in the used session types. This would result in a trace that would be as complicated as one that is the result of two or more managers, or SNMP enabled network elements for that matter, operating from the same IP address, where more than one generates sessions that are of the same session type. As a consequence, the least often occurring session types are being considered first.

Although this may seem sufficient to find all core session types, there is still a problem. In small polling instances, all session type occurrences can be found relatively close to each other, in which case it would be easy to find all core session types. But, in the case where sessions can take very long in time and occur sequentially only and may also be very large, occurrences of related session types, may be removed from each other more than the stated timeout. Therefore, some related session types may incorrectly be considered as not being a session type contributing to a set of potential session sets that are of a session set type. This problem will be addressed in the subsequent points.

Some of the twelve session types can be considered as core session types:

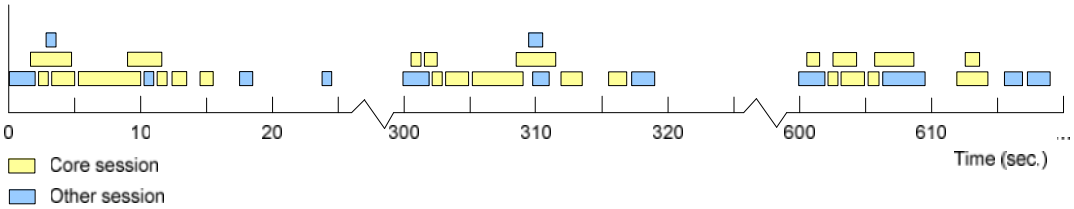

Figure 4.7: Some sessions are of a core session type

Although it would be expected that all twelve session types are considered a core session type, it may be the case that at a point later on in the trace, which is not shown, the occurrences of related session types were too far removed from each other in time. As a result, only 8 of the 12 session types have at this point been considered as core session types. The sessions of these session types are marked in figure 4.7.

5. Scan for other potential session set members

Already the general structure of the potential session sets, making up a single session set type becomes clear in the example. But, there are still sessions of session types that should logically be considered part of a potential session set. These sessions need to be detected.

This step attempts to find the irregularly occurring session type occurrences, or the occurrences of related session types, by taking the now existing preliminary session sets, which only consist of core session type occurrences, and subsequently looking for sessions occurring within the stated timeout value of the existing core session type occurrences.

Every preliminary session set that exists at this moment shall be considered one at a time. When one preliminary session set is considered, then for every session that occurs in that session set so far shall a scan be made that looks for sessions occurring within the timeout value of that session. Then, every session that occurs within this range and is not yet listed as either a core session type occurrence, or potential session set member, shall be taken into consideration. In some cases, it may also be possible to exclude potential sessions that are of a session type of which already a session is detected in a particular preliminary session set.

This step can be visualized in the abstract example as follows:

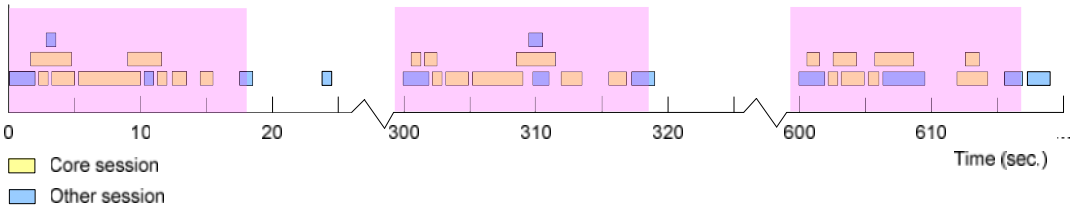

Figure 4.8: Selected scanning ranges from existing sessions

The pink marker field shows the time range that is at all times within the stated example timeout of 3 seconds from any of the sessions that are found so far. As can be seen, some sessions are now within this range and are not yet listed as potential session set members. When these other sessions are considered as potential session set members, this results in the following graph:

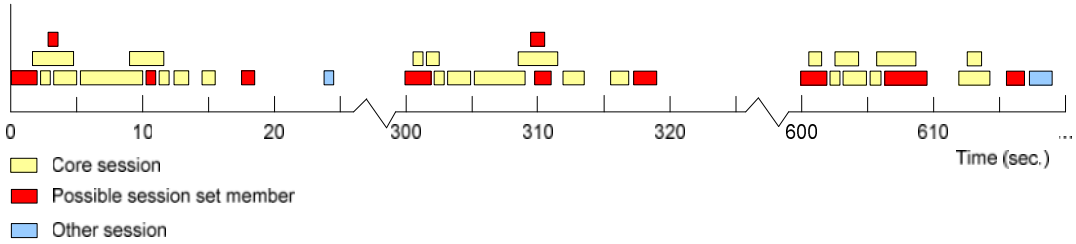

Figure 4.9: Possible new session set members selected

The next step is to repeat this process for the now new members of the respective potential session sets. This will go on until either an occurrence of an already listed session type is scanned (if this filter is selected), or when no new sessions can be found as a result of the fact that the time till the next session is greater than the stated timeout value.

Only one other scan shall take place after the first one in the case of the abstract example:

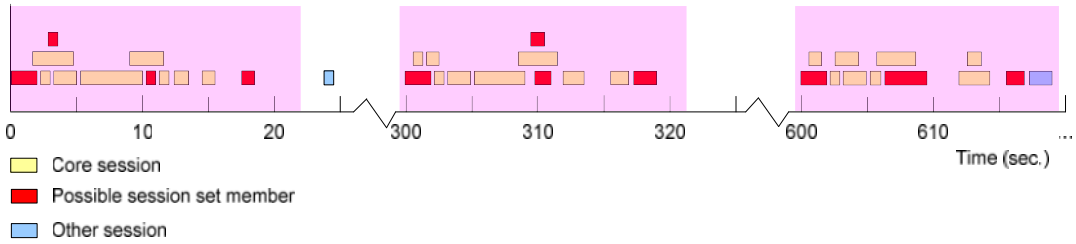

Figure 4.10: Selected scanning ranges from existing sessions

This figure shows that all occurrences of the twelve session types are now selected for the next step. Before this last scan, the third potential session set missed one session type occurrence, which has now been found. Also, note that the scanning ranges are now a bit larger than before, because there are more sessions in the potential session sets around which the range of 3 seconds is considered. Because the selected sessions in the potential session sets all occur within the timeout range of each other, are all three shown scan ranges forming a single block without wholes respectively.

After this last scan, the abstract example looks like this:

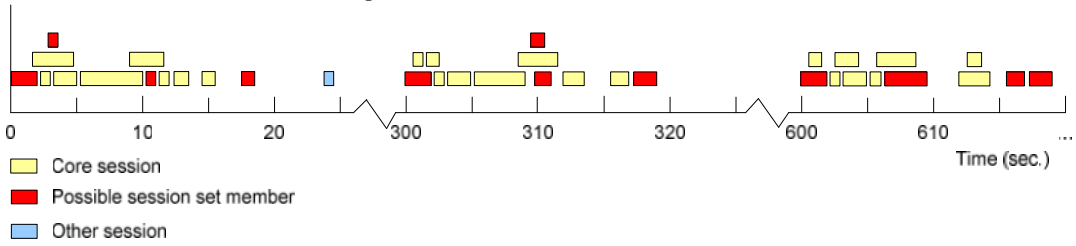

Figure 4.11: Potential session sets after completing the scanning process

Observe that just one session was not considered to be part of any of the potential session sets, because it occurred more than the stated timeout value from any of the members of any potential session set. Also note that of the three potential session sets shown, all twelve session type occurrences are found. These potential session sets shall now be taken to the next step.

Note: in this example it is chosen to allow only one occurrence of a session type in every session set, in other scenarios it may be desirable to allow multiple occurrences of a single session type within a single session set. In such a case, only the timeout value shall restrict the scanning process. Based on some manual inspections of various trace files, it is clear that in most cases it will not be desirable to allow multiple occurrences of a singe session type within a single session set.

6. Detect proper session types

The potential session sets that have been found so far may contain session types that do not occur often enough to be considered part of the session sets of a single session set type. As has been discussed in the previous subsection, only session types that occur often enough (in enough potential session sets), shall be considered part of the session sets of a particular session set type. This step will therefore look for session types that do occur often enough.

This step does so by starting with the session type that occurred most often in the potential session sets known so far. Then it takes the second session type and looks to see if these two both still occur often enough in potential session sets. This continues until it attempts to add a session type to the test set of session types, which results in too little session sets containing an occurrence of each of these session types. Then, it will try the same with a different session type, until it finds the largest possible set of session types of which each of these session types is occurring in enough session sets. When one session type is not satisfactory at some point in this process, then it may be reconsidered at a later point in this process. In the end the largest possible group of session types shall be found. The smallest group of session types consists of the core session types.

In the case of the abstract example, there are twelve session types. These are all part of regular polling behaviour and shall therefore be found in all session sets. The other mentioned session type is not part of any of the potential session sets at this point, so only twelve different session types can be found in these session sets. Since the three session sets shown are just the first three of an artificial trace, there may at the end of this step be 100 detected session sets. As mentioned earlier in this example, the minimum occurrence is set to 80%. Thus, the largest possible group of session types should be found in at least 80 of the 100 session sets found. In this step, the algorithm will start by taking one of these twelve session types, then another, until it finds the largest group of session types possible. Since this is a very simple example, all twelve session types will make up this largest group of session types, because they occur in all 100 session sets. Therefore, these twelve session types will be taken to the next step.

7. Determine session sets of the same session set type

In this step shall for each of those potential session sets which were found, those sessions be selected that are of one of the selected session types. Then, in these session sets all occurrences of session types that are not a member of the selected group shall be discarded, which means that they will be considered in another instance of this algorithm. This step will result in session sets that equal to or smaller in size than the previously detected potential session sets, which now only contain occurrences of the session types that were selected in step 6.

The selected session sets are all of a new session set type, if the number of session sets is larger that the limit of minimum session sets per session set type (which is equal to the limit of minimal number of session type occurrences of a session type in the session sets of a particular session set type). After this, all involved sessions that make up these session sets shall be removed from the trace. If the minimum number of session sets is too small, then all session sets shall be added to a list of *floating session sets* (session sets that are not assigned to a session set type), which can result in a different or separate session set type later on in this algorithm.

If there are still some potential session sets remaining after removing the selected session sets from the inventory, then they shall be considered as follows:

- They will all be considered as floating session sets, if the total number of these remaining session sets is smaller than the limit of minimum session sets per session set type. The respective session sets and their sessions shall subsequently be removed from the inventory;
- If there are still more potential session sets than the stated limit, then the process, with these potential session sets, shall go back to step 6, until either the number of session sets is too small, or there are no potential session sets remaining after step 7 has been considered again.

The algorithm returns to step 3, if there are no - or less than the stated limit - remaining potential session sets at this point and if there are still other unhandled sessions initiated by this initiator. If this is not the case, the algorithm shall go to step 8.

In the case of the example, shall all session sets remain unchanged, because all session sets only contain occurrences of session types that are a member of the selected twelve session types of the group of the previous step. The resulting session sets, which are of a single session set type, can be shown as follows:

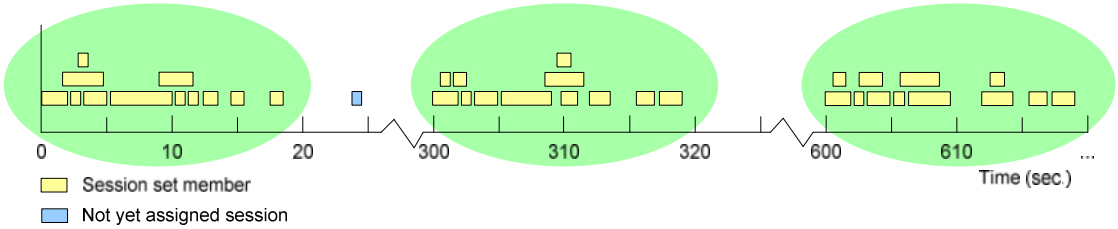

Figure 4.12: Session sets of a single session set type found in the first ~600 seconds of the trace

After this, there shall remain only a single session, as depicted in figure 4.12. The algorithm shall consider this single session as a single session set, of just one session. Because the set of session sets is then smaller than the stated limit, the session set shall be added to the list of floating session sets.

8. Handle floating session sets

This step involves the possibly generated floating session sets. This step will try to find a session set type of which a floating session set may be part, or it will create a new session set type and assign a floating session set to it.

The assignment of a session set to an existing session set type occurs only if the following conditions are met:

- They will all be considered as floating session sets if the total number of these remaining session sets is smaller than the limit of minimum session sets per session set type. Also in this case shall the sessions making up the floating instances be removed from the inventory;
- The floating session set must have a significant match of occurring session types compared to the longest session set of a session set type.

If these conditions are met, then a floating session set shall be added to such an existing session set type. If not, a new session set type shall be created consisting of only that floating session set. After that, the following floating session set shall be considered, until all have been either assigned to an existing session set type or have resulted in the creation of a new session set type.

The next activity within this step is the attempt to merge session sets that should be considered one. Merging will only occur if the matching of the occurring session types is significant. This measure has been added, because of the possibility that separate session set types are being created as a result of floating session types.

In the end, all sessions initiated by a specific initiator shall have been considered, after which the algorithm goes back to step 2, if there are still other sessions belonging to different initiators. If not, the algorithm goes to step 9: the last step.

In the case of the abstract example, only the single separate session, which did not belong to any session set yet, resulted in a floating session set. In this step, this floating session set would have been considered as part of a new session set type, because it would not match with the only created session set type and its session sets. Also, that session set would be of a length that is not within the stated limits of 50%; i.e. the minimum length for a possible session set assignment would be 50% of twelve. But, this session set only contains one session type occurrence.

9. Finishing

This is the final step of this algorithm that will be reached when it has handled every session that was given to it as input (minus some possible border sessions). This step will present the results to the next algorithm as independent session set types, involving session sets in accordance with the given definitions of the two terms.

In the case of the abstract example, two session set types would have been detected, using the given preset limits. One of the session set types would encompass all session sets which were initiated by the manager as a result of pre-programmed regular polling behaviour. The remaining session, which was completely unrelated to the session sets of the first session set type, resulted in a different session set type:

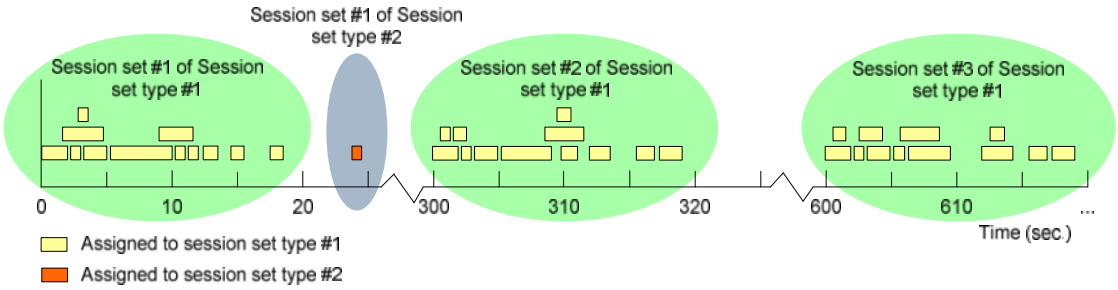

Figure 4.13: Resulting allocations of the given sessions

4.4.2 Example algorithm execution

Although the description of the algorithm and its steps has been given alongside an example, that example was very abstract and straightforward. A more elaborate example is given in appendix A3.

Chapter 5

Determination of periodicity and interval detection

The definitions and algorithms of sessions, session types, session sets and session set types have now been discussed. The traffic identification problem, as stated in chapter 1.2, has now been addressed, since SNMP messages that are related to each other are now grouped into session sets. Moreover, session sets that have a significant level of commonality with each other are of the same session set types. But, now the question remains how can it be determined whether these session sets are periodic or aperiodic? Also, in the case of periodic session sets, what are the intervals that can be found between these session sets? This chapter will discuss the algorithm that has been developed to solve these questions.

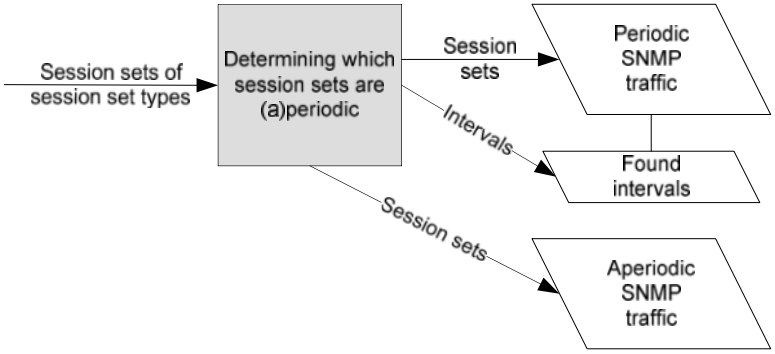

Figure 5.1: Input and output of this algorithm

As shown in figure 5.1, this algorithm will accept session sets of a session set type and return whether a session set is periodic or aperiodic. Also, in the case of periodic session sets, it will return the discovered intervals.

In order to give a better understanding of the problems regarding this algorithm, this chapter is separated into the following sections:

- An introductory example shall be discussed briefly in order to identify just some of the problems and characteristics of both kinds of session sets;
- Then, before anything can be stated about the algorithm, it is desirable to discuss the exact characteristics of both periodic and aperiodic session sets;
- Based on these consideration, it will be possible to give a definition of both kinds of session sets;
- Then, the algorithm shall be discussed, which shall be followed by an elaborate example.

5.1 Introductory example

Although the terms periodic session set type and aperiodic session set type seem straightforward, they both do raise the question about when a session set type is supposed to be considered either one or the other of these. Two example session set types shall be shown here to highlight this problem.

The first session set type is shown here and involves six session sets that are of this session set type:

Start Time (sec.)	Session set
50	Session set #1
200	Session set #2
350	Session set #3
500	Session set #4
650	Session set #5
800	Session set #6

Table 5.1: Sessions sets making up a single session set type

This session set type shows that there are six session sets that are of this single session set type. The time between the start points of the various session sets suggests that there is a clear interval of 150 seconds between every start point. In such a case, it would be easy to conclude that this session set type is a periodic session set type, because all of the six session sets are contributing to a specific interval.

The following table shows a different session set type and its session sets:

Start Time (sec.)	Session set
20	Session set #1
180	Session set #2
320	Session set #3
400	Session set #4
550	Session set #5
780	Session set #6

Table 5.2: Sessions sets making up a single session set type

Again, this is a session set type that contains six session sets. But, in contrast with the previous example, this example is less clear. By looking at the time of the various start points, it would be tempting to conclude that there is a strict pattern for some of the session sets. For instance, the time between the start points of session set #1, session set #2 and session set #3 suggest that there is almost an interval of around 150 seconds. But in the first case, it is 160 seconds and in the second it is 140 seconds. Then, there is also the issue of the fourth session set, which occurs just 80 seconds after the third session set started. This raises the question: should this session set type still be considered periodic, or just a portion of the session sets of this session set type?

The following basic characteristics of both kinds of session set types can be summarized as follows:

Periodic session set type:

- There is a clear single interval between the occurrences of the respective session sets.

Aperiodic session set type:

- There is NO clear single interval between the occurrences of the respective session sets.

5.2 Considerations

The basic characteristics and problems regarding the determination of periodic or aperiodic behaviour of session sets have already been discussed briefly. This section considers a number of problems that may occur in the real world, which have to be dealt with in the algorithm. Also, the eventual complete definitions of both kinds of session set types shall be given, based on these considerations.

5.2.1 Timer related issues

Assume that a manager is programmed to poll a set of agents every 300 seconds. Then, the traffic belonging to a particular polling instance shall be grouped into a session set, which shall be part of a session set type. But, at which point does the manager reset its timer and wait for the next polling instance to be started? The following overview of the member sessions of a session set shows the possible reset points:

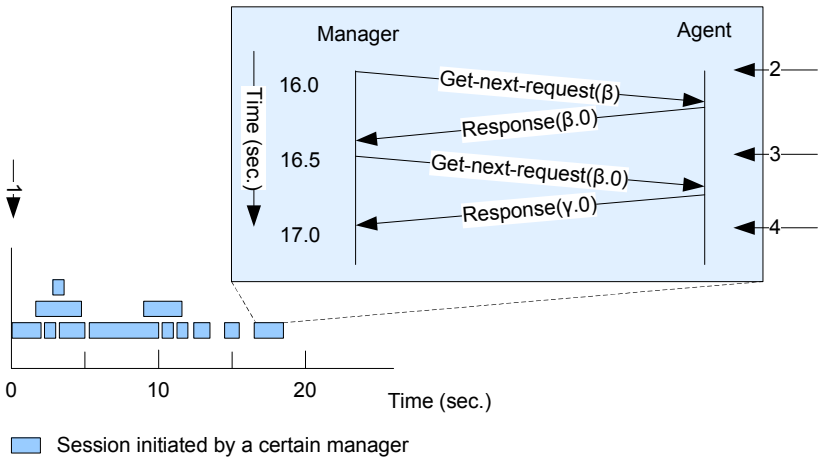

Figure 5.2: Possible timer reset points

This figure shows 12 sessions that make up a single session set. If, for example, a manager is set to poll a number of agents, where this figure shows the resulting sessions from a single polling instance, then there are the following possible timer reset points on which the manager determines to wait 300 seconds, before it starts the next polling instance:

1. The moment at which it starts with the first initiator message of the first session belonging to a polling instance;
2. The moment of the first request for the last agent to be polled (the last session of a polling instance);
3. The moment of the last request for the last agent to be polled (the last request being sent, before the manager has completed a polling instance);
4. After the manager has received the last response to the very last request of the last session, as part of a polling instance.

The determination of the reset point is an important process, because if this is not done properly, an incorrect conclusion may be made regarding the periodicity of a session set type and its session sets. For instance, if a manager resets its waiting timer to 300 seconds in point 4, then the unsuspecting observer may state that there is no clear interval between the session sets. The following graph shows such a situation:

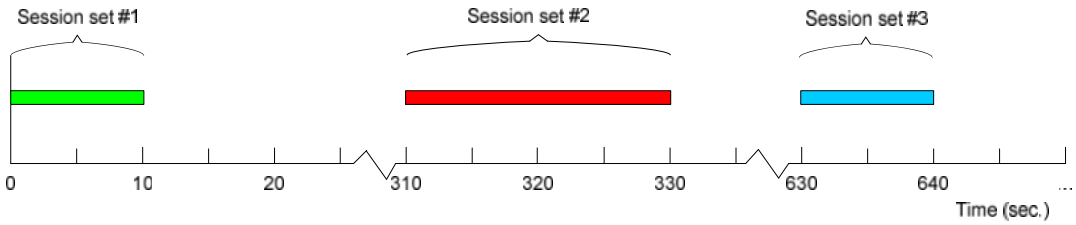

Figure 5.3: Polling with timer reset point 4

This figure shows that when the interval between session sets of a session set type is considered based only on the start points of the session sets, then in this case there will be an interval of 310 seconds between the first and the second and an interval of 320 seconds between the second and the third. This is possible, since

session sets can sometimes take longer than other session sets of the same session set type. Still, in this case there would be a clear problem, because with an interval that deviates so much from an average interval, it is questionable to conclude that this session set type is to be considered periodic. If, on the other hand a mechanism is developed, which can detect the proper timer reset point, then these difficult situations will arise less often. However, such a mechanism may require a large number of session sets, in order to determine the proper reset point.

Even when the timer reset point is determined correctly, or known ahead, there is still room for deviation in the exact start time of, for example, a polling instance. Some possible causes for this are the following:

- A trace is recorded on some point of a network. This means that there is always some delay between the time an SNMP message is sent and the time it is seen and recorded at the recording unit. This means that the exact start point, from the perspective of the recording unit, may deviate up to a few seconds;
- The waiting method used by, for example, a manager between polling instances is usually based on a mechanism that is not very accurate. For instance, when a manager application uses the sleep() method in Perl, like MRTG [7], then the actual interval between polling instance may be up to one second removed from the desired waiting period. This is the result of the fact that this method only accepts an integer amount of seconds.

Clearly, there are numerous aspects regarding the timer mechanism used for polling instances, or at least periodic behaviour. The following points have been highlighted in this subsection, which need to be incorporated in the algorithm:

- Different SNMP traffic initiators, which generate SNMP traffic on a regular basis, may use different timer reset points. It is paramount that the proper reset point of a SNMP traffic initiator is detected, before anything can be stated about the periodicity of the session sets of a session set type;
- There is usually some time (up to a few seconds) between the moment of sending an SNMP message at a network element and the moment at which this message is recorded at the recording unit. This makes it also more difficult to exactly determine an interval;
- The waiting mechanism used by SNMP traffic initiators between periodic activities could be inaccurate, which could result in a deviation from the desired waiting time between these periodic activities.

5.2.2 Multiple intervals within a session set type

Up to now, only periodic session set types, which have a single interval between their session sets, have been considered. However, in the real world, there could be a possibility that a periodic session set type contains multiple intervals between its session sets. The following shows an example session set type, which has two intervals: 150 seconds and 300 seconds.

Start Time (sec.)	**Session set**
10	Session set #1
100	Session set #2
160	Session set #3
310	Session set #4
400	Session set #5
460	Session set #6
610	Session set #7
700	Session set #8
760	Session set #9

Table 5.3: Session sets making up a single periodic session set type with two intervals

This table of session sets shows the difficulty of determining which interval(s) can be found in a periodic session set type. Also, it is usually not known in advance which interval(s) were used in practice. If this possibility of multiple intervals is not considered, then this algorithm may in some cases incorrectly conclude that this session set type is aperiodic. Therefore, the algorithm needs to be able to identify all intervals between the session sets of a session set type.

This subsection has stressed the following:

- Within a single session set type may be session sets contributing to multiple intervals, which are used to generate periodic traffic. If not all intervals are found correctly, an incorrect conclusion could follow which would state that such a session set type is aperiodic.

5.2.3 Composite of periodic and aperiodic behaviour

Besides multiple intervals within a single session set type, it may even be possible to have session set types that have, besides just aperiodic or periodic, also periodic or aperiodic session sets respectively. This could result in significant problems and may also lead to incorrect conclusions regarding certain session set types. The following example session set type shows this possibility.

Start Time (sec.)	Session set
10	Session set #1
310	Session set #2
610	Session set #3
655	Session set #4
910	Session set #5
1210	Session set #6
1510	Session set #7

Table 5.4: Session sets making up a single session set type containing both kinds of session sets

This table shows that there are 7 session sets, of which most of them are following each other in a regular fashion, such that an interval of 300 seconds could be detected. However, there is one session set that does not seem to follow this regular pattern: session set #4. If this session set would be disregarded for a moment, then all session sets follow 300 seconds after the previous one. This would remain session set #4, which seems to be of a different interval, or is just an aperiodic occurrence. It could be the result of an aperiodic activity, caused by, for example, the manually initiated polling of the same set of agents by a specific manager.

This section has stressed the following:

- Session set types may contain both periodic and aperiodic session sets. The periodic session sets shall act in a regular expectable fashion, based on (a) specific interval(s). The aperiodic session sets do not act in a regularly detectable and expectable pattern.

5.2.4 Trace holes

Another issue that could occur, due to various reasons, are holes in a trace, which would result in unexpectedly large chronological gaps between the occurrences of the various session sets that make up a session set type. The following table shows such a possibility.

Start Time (sec.)	Session set
35	Session set #1
335	Session set #2
635	Session set #3
1180	Session set #4
1480	Session set #5
1780	Session set #6
2080	Session set #7

Table 5.5: Session sets making up a single session set type containing a hole

This table shows a hole in the table between the third and fourth session set of a session set type. Observe that the first three session sets seem to be part of a periodic behaviour, based on an interval of 300 seconds. The same can be stated about session sets 4 through 7. So, do both parts actually belong together?

A few possible reasons for observable holes in a table are:

- The recording of traffic at the recording unit got interrupted for a certain amount of time;
- The network element responsible for initiating the session sets of this session set type stopped its regular behaviour for a certain amount of time, but resumed at a later point, using the same interval as before the interruption.

This subsection has shown the possibility of observable holes in a trace. Therefore, the following can be concluded on this topic:

- Although a network element may be programmed to perform perfectly expectable patterns of regularly occurring session sets, there may always be holes, due to various causes, that would lead to unexpected occurrences of session sets (i.e. at different times than would be expected, based on the knowledge of an interval). This makes it more difficult to determine which session sets contribute to a specific interval within a session set type.

5.2.5 Border session sets

An issue that has also been considered in the case of the previous algorithm, but should again be considered here, is incomplete session sets at the beginning and at the end of a trace.

Even though incomplete sessions have already been taken care of, it is still possible that the recording started or stopped halfway an active polling instance, or another regularly occurring session set. Hence, it is necessary that these incomplete session sets are removed from a session set type, before they are considered for the periodicity analysis. The following example will demonstrate the necessity.

Start Time (sec.)	Number of sessions in session set	Session set
59	5	Session set #1
350	10	Session set #2
650	10	Session set #3
950	10	Session set #4
1250	10	Session set #5
1550	10	Session set #6
1850	6	Session set #7

Table 5.6: Session sets making up a single session set type with incomplete border session sets

The first listed session set contains just 5 sessions, where the ‘normal’ session sets contain 10. This would suggest that some sessions have not been recorded. As a result, the observed starting time of that session set cannot be determined correctly, since the actual first session of this session set is not recorded. If this

session set would be considered as a regular session set of this session set type for processing, then it may be incorrectly concluded that the first session set does not contribute to the periodic behaviour with a 300-second interval.

This logic can also be applied to the very last session set. However, in this case it is not the start time, but the end time of the session set that is not recorded. This could affect the results of the timer reset point determination process. A later discussion on this topic will stress that the end time of the last (incomplete) session set is not relevant, but only the start time of the last session set.

The following can be concluded, based on the observations made in this subsection:

- Incomplete session sets, which occur at the very beginning or at the end of a trace and are of a specific session set type, ought not to be considered in the process of determining periodicity and should therefore be removed beforehand.

5.3 Definitions

In the introductory chapter, the two terms *periodic session set type* and *aperiodic session set type* have already been discussed briefly. Based on the considerations stated in the previous subsection, it will be required to add a third session set type: a *composite session set type*, because there is also the possibility of a combination of both periodic and aperiodic session sets within a single session set type. Following are the three definitions and the respective characteristics of these three terms, which have been based on the considerations discussed in the previous subsection.

Before anything can be stated about the exact definitions of the three types of session set types, it is desirable to explain what is exactly understood by a *periodic session set* and an *aperiodic session set*. Following are the definitions used for these two terms:

Definition A *periodic session set* is a session set that is part of a subset of session sets that are all of a specific session set type, which all occurred based on a specific definable single interval.

Definition An *aperiodic session set* is a session set that is NOT part of a subset of session sets that are all of a specific session set type, which all occurred based on a specific definable single interval.

These two and the following definitions have been based on the topics described in the previous subsection. The meaning of them shall be used for the remainder of this thesis. Following are the definitions of the three variations of session set types.

Definition A *periodic session set type* is a session set type which has the following characteristics:

- One or more intervals can be found between the session sets;
- All session sets (possibly without an incomplete first session set) contribute to one specific interval based behaviour only (not to more than one). Thus, each session set is a member of one subset that contains session sets that are contributing to the same interval;
- No aperiodic session sets may exist.

Definition An *aperiodic session set type* is a session set type which has the following characteristics:

- No subset of session sets contribute to a specific definable interval;
- All session sets are marked aperiodic.

Definition A *composite session set type* is a session set type which has the following characteristics:

- Some of the session sets are part of a set of the session sets that contribute to a specific definable interval;
- Some of the session sets do not contribute to a specific interval, but are purely aperiodic.

5.4 Algorithm

The definitions of the involved terminology and the most significant problem scenarios regarding this algorithm have been discussed. This algorithm takes as input session sets that are of one specific session set type and returns the periodic session sets and aperiodic session sets (if available). When a few session sets result in a detectable behaviour with a specific interval, then such a set of session sets shall be considered a subset of the session sets that contribute to one specific interval. An abstract example description is given alongside the algorithm description, in order to clarify the steps taken in this algorithm. A more elaborate example shall be given after that description.

5.4.1 Algorithm description

This algorithm is developed, based on the definitions and considerations given in the previous subsections of this chapter. Following is an extensive description of this algorithm, which will be accompanied by a very simple example, in order to clarify the steps taken, which are shown in the following flow diagram.

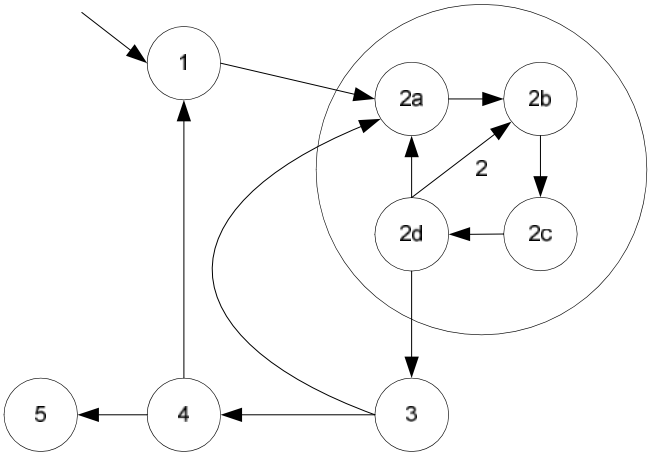

Figure 5.4: The steps taken in this algorithm

The core idea, around which this algorithm is developed, is the finding of the smallest intervals in session set types first, before the larger ones. This is especially important in cases where session set types contain multiple intervals, like for example 100 and 300 seconds. When first the larger intervals are detected, it could be possible that the smaller intervals become invalidated and undetectable, if those session sets contributing to larger intervals are removed after finding them. If no reasonable interval can be found for a particular session set, then it will be considered aperiodic. Finally, this algorithm uses the assumption that a single session set is either of the following within a session set type: an incomplete start session set, in which it will be marked as such, aperiodic, or it is periodic in which case it may contribute to only one interval.

As was the case with the previous algorithm, this algorithm also requires the definition of some predefined values. These are usually trace specific, still they can be set by estimating and making consideration. The following table shows the limits or variables that have to be set, before the algorithm is executed.

Limit	Value
Max. start session set length difference	The maximum length difference, compared to the average session set length, of the first session set of a session set type.
Max. interval deviation	Within a subset of the session sets that occur with a specific interval, the maximum deviation from the average interval between the session sets shall be this amount.
Max. average interval deviation	Within a subset of the session sets that occur with a specific interval, shall this limit be the maximum average deviation from that interval.
Min. nr. of session sets in subset	The minimum number of session sets within a set of session sets that contribute to the same interval.
Max. time between session sets	The maximum time between any two session sets within a set of session sets that contribute to the same interval.

Table 5.7: Definition of preset limits/variables used in this algorithm

The second and third values avoid large changes in the intervals between the session sets that are considered to be contributing to a single interval. The last one is used to detect time gaps between session sets. The following values shall be used for these limits in the example described alongside the taken steps:

Limit	Value
Max. start session set length difference	25%
Max. interval deviation	30 seconds
Max. avg. interval deviation	10 seconds
Min. nr. of session sets in subset	3
Max. time between session sets	650 seconds

Table 5.8: Preset values used in the explanatory example

Following is an example of a session set type, which could be given as input to this algorithm. This session set type shall be used to explain the steps taken in this algorithm.

Start Time (sec.)	Last session start (sec.)	Last request (sec.)	Last response (sec.)	Number of sessions in session set	Session set
0,23	7,27	7,32	7,46	12	Session set #1
150,50	159,10	159,88	159,98	20	Session set #2
300,40	308,12	308,39	308,42	20	Session set #3
449,93	457,15	457,78	457,95	20	Session set #4
600,98	609,02	609,22	610,34	20	Session set #5
750,11	758,57	758,67	758,72	20	Session set #6

Table 5.9: Session sets making up the example session set type

A first glance at this table of session sets suggests that there is little periodicity to be determined in this session set type. Following are the steps that make up the algorithm that will yield a clear conclusion on this session set type and its session sets.

1. Remove border session sets

The necessity of this step has already been discussed in chapter 5.2. This step involves the removal of the first session set, if this one seems incomplete. The last session set, as in the previous algorithm, is not eligible for removal, since this algorithm only looks at the intervals, which for the last session set is the time at which this last session set starts, not when it ends. If a session set is considered to be part of a session set type in the previous step, then there is no need to recheck this validity.

First, the average session set length is determined, without any of the two border session sets. If three or less session sets are available, then the largest session set shall be considered for the average length value. Then, this average length value is multiplied by (1-*Max. start session set length difference*), yielding the minimum number of sessions in the first session set. If the first session set is smaller than that value, it shall be removed, before the algorithm goes to the next step.

In this example, the calculated average session set length would be 20 and (1-0.25)*20 = 15. But, since 12 is smaller than 15, the first session set shall be considered incomplete and it will be discarded.

2. Detect set of session sets with the same interval

This step is the most important and significant one of this algorithm. Here, a set of session sets is to be detected that contribute to the same interval, if that can be found. The description of this step has been split into four sub steps, in order to keep this second step understandable.

2a. Find first two session sets

The first thing done in this step, is the detection of the first session set that has not yet been assigned to a set of session sets with a specific interval. Then, the next unassigned session set is looked for. If this next session set is found and it occurs within *Max. time between session sets*, then the interval between these two session sets is determined. If this requirement is not met, then this first single session set shall be considered aperiodic and it shall be removed from the input listing.

As a result, the first and the second session set shall simply be selected for consideration. If that does not result in a large enough set of session sets to be considered contributing to a single interval, then when the algorithm returns to this step after step 2d, it will attempt to keep the first session set selected and it chooses the third session set as second session set for consideration. This goes on with the same first session set selected, until either a large enough set is found in step 2d, or the gap between the first and second selected session set is larger than *Max. time between session sets*. In this last case, the first session set shall be considered as aperiodic and subsequently a new first session set shall be selected and the process shall be repeated.

2b. Determine interval between selected session sets

The interval determination of a set of session sets involves two aspects. One is the determination of the intervals between the session sets, measured for the four timer reset points. The other is the determination of the proper timer reset point. Since a periodically repeating pattern shall use a specific interval, alongside a specific timer reset point, it is expected that in the case of the timer reset point the total deviation from the interval measured from that specific timer reset point is the smallest compared to the other timer reset points. This does require enough intervals, in order to calculate the correct timer reset point.

First, the intervals for the four cases, as shown below, need to be calculated.

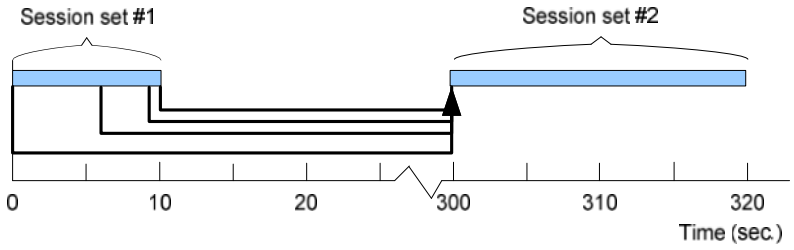

Figure 5.5: Four interval cases

The intervals between *n* listed session sets will be calculated as follows:

Interval calculation using reset point *r* (r = 0:3):

$$\text{intervalSum} = \sum_{i=0}^{n-1} (\text{getTimeAtResetPoint0}(\text{sessionSet}[i+1]) - \text{getTimeAtResetPoint}R(\text{sessionSet}[i]))$$

$$\text{interval} = \text{intervalSum} / (n - 1)$$

If there are more than two session sets involved, then the average interval shall be calculated and also the total deviation from this particular interval, which is calculated as follows:

Deviation calculation using reset point *r* (r = 0:3):

$$\text{deviation} = \sum_{i=0}^{n-1} |\text{averageIntervalForThisResetPoint} - (\text{getTimeAtResetPoint0}(\text{sessionSet}[i+1]) - \text{getTimeAtResetPoint}R(\text{sessionSet}[i]))|$$

After this has been done is it possible to state which reset point should be considered and thus which interval shall be used. This will be the interval where the total deviation from the respective interval is the smallest. It must be noted that this process will become more accurate when there are more session sets to be considered. In the case of just two session sets, the reset point shall be chosen automatically at the start of a session set, since manual trace inspections suggest that this reset point is used most often. This predefined choice is necessary, because in the case of just two session sets, the deviation from the average interval is for every reset point zero.

2c. Determine range of next session set

The next activity, after the interval determination, is the detection of the next session set by defining a range, based on the interval between the existing session sets. This is shown in the following figure.

Figure 5.6: Finding session sets with a single interval

The range is calculated, based on the interval between the currently considered session sets of this interval and the values *Max. interval deviation* and *Max. avg. interval deviation*, which were preset at the beginning. This range determination process does not make use of the information of any other session set than those listed in the set of session sets so far. Based on the interval of this set of session sets, it is possible to expect the next occurrence within a certain range. The calculation goes as follows:

maxNextIntervalDeviation = (#{found session sets with this interval} * (Max. avg. interval deviation))-

(Total deviation so far)

maxNextIntervalDeviation = MIN(maxNextIntervalDeviation, (Max. interval deviation))

lowerRangeLimit = (Time at reset point of last listed session set) + interval - maxNextIntervalDeviation

upperRangeLimit = (Time at reset point of last listed session set) + interval + maxNextIntervalDeviation

As has been noted in the previous sub step, in the case of just two session sets based on which an interval and range is estimated, it should be noted that the range of the expected next occurrence is very large as a result of the preset values. This is because of the fact that there is still no deviation from an interval, thus at this point it can still be very large.

2d. Detect the next session set for this interval

Once the range is known, it is possible to search for a session set of the given session set type that occurs within this range. One of the following is possible:

- If this is the case, then this session set shall be added to the subset for this interval and the algorithm shall return to step 2b;
- If there are more than one session sets occurring within the set range, then the session set that is deviating the least from the interval shall be chosen and the algorithm returns to step 2b;
- If there are no session sets within this range and
    - The subset of session sets is still smaller than the limit (*Min. nr. of session sets in subset*), then another first pair shall be tried in step 2a;
    - The subset is larger than or as large as the limit (*Min. nr. of session sets in subset*), then the algorithm goes to step 3.

In order to clarify the operation of this algorithm in these sub steps of this algorithm, here follows the application of step 2 on the example input.

The first listed session set was removed, because it was considered incomplete in step 1. Then, the first session set is started at 150,50 seconds. Here follow the steps taken:

- In step 2a also the second session set shall be found, because it is within the stated limit of 650 seconds. This will make the following two session sets eligible for consideration of a single interval;

Start Time (sec.)	**Last session start (sec.)**	**Last request (sec.)**	**Last response (sec.)**	**Number of sessions in session set**	**Session set**
150,50	159,10	159,88	159,98	20	Session set #2
300,40	308,12	308,39	308,42	20	Session set #3

Table 5.10: Session sets that apparently contribute to a single interval

- In step 2b the intervals for every reset point shall be determined. This gives the following:

Initiator message of first session (sec.)	**Initiator message of last session (sec.)**	**Last request of last session (sec.)**	**Last response of last session (sec.)**
148,90	141,30	140,52	140,42

Table 5.11: Detected intervals for the given set of two session sets

  The interval based on the first initiator message shall be chosen for the next step, since there is just one interval and all deviations from the respective intervals are zero;
- Step 2c determines the range in which the algorithm expects the next session set to occur with this interval. The calculation for the range is as follows:

$$\text{maxNextIntervalDeviation} = (2 * 10) - (0) = 20$$

$$\text{maxNextIntervalDeviation} = MIN(20, 30) = 20$$

$$\text{lowerRangeLimit} = 300{,}40 + 148{,}90 - 20 = 429{,}30$$

$$\text{upperRangeLimit} = 300{,}40 + 148{,}90 + 20 = 469{,}30$$

As a result, a session set is expected to start between 429,30 seconds and 469,30 seconds in trace time;

- The example shows that there is indeed a session set within the stated range, and step 2d will therefore add this third session set to the subset of session sets that occur, based on an average of around 150 seconds. Subsequently the algorithm returns to step 2b, where it will recalculate the interval;
- The algorithm will go from step 2b to 2c, in which it will detect the next session set. This process shall go on, until it reaches the end of the list of session sets in this session set type. In the end, there will be six listed session sets within the created subset of session sets that contribute to this single interval. Then, the algorithm shall go to step 3, because this subset contains more than the minimum amount of session sets, as indicated by the respective limit.

At this point the algorithm has detected the following set of session sets that all seem to contribute to a single interval and shall be taken to step 3 by this algorithm.

Start Time (sec.)	**Last session start (sec.)**	**Last request (sec.)**	**Last response (sec.)**	**Number of sessions in session set**	**Session set**
150,50	159,10	159,88	159,98	20	Session set #2
300,40	308,12	308,39	308,42	20	Session set #3
449,93	457,15	457,78	457,95	20	Session set #4
600,98	609,02	609,22	610,34	20	Session set #5
750,11	758,57	758,67	758,72	20	Session set #6

Table 5.12: Set of session sets taken to step 3

3. Store the created set of session sets with the same interval

When the algorithm reaches this step, it means that it has found a set of session sets that behave in accordance with a specific interval. It is the purpose of this step to store this set of session sets accordingly.

When no other set of session sets of a particular interval has yet been saved, then the set given to this step shall just be saved for later on, when it will be part of the results of this algorithm. However, it could be possible that there is already a set saved.

In the latter case, every already existing set shall be tested to see whether the latest set may contain session sets that behave with the same interval as a previously saved set. Two or more sets with the same interval may be found as a result of, for example, large unexpected gaps between two session sets of a session set type. In order to test for this equality in interval, all session sets of the first found set shall be added to the session sets of the latest set. Then, the algorithm will determine whether the intervals and deviations, are within the preset limits. If this is the case, then the latest set of session sets shall simply be added to the already existing set of session sets. This measure means that sets with an average interval of, say, 152 seconds and a different set with an average interval of 149 seconds can be combined. It should be added that any session set that is marked aperiodic shall not be compared with. If, however, no mergeable set could be found, then this latest set of session sets shall simply be stored separately from those.

The algorithm will continue in step 2, but now with the next session set of the given session set type that has not yet been assigned. If, on the other hand no remaining session sets can be found, then the algorithm goes to step 4, where it will finalize the results.

4. Finishing

This is the last step taken by this algorithm for a session set type. This is simply the endpoint of the algorithm for this session set type, in which it will release the results of the session sets of this session set type. This result will include all found intervals and the session sets that are responsible for them respectively. But, also the aperiodic session sets which could not be assigned to a set of session sets that contribute to a particular interval. If there is still more input to this algorithm, then the algorithm will start in step 1 with the new session set type, or otherwise terminate in step 5.

In the case of the discussed example, only one interval can be found with the set of session sets shown in table 5.9. The result of this step for that particular set includes the following information per timer reset point:

Timer reset point	Interval (sec.)	Total deviation (sec.)
First initiator of first session of session set	149,90	2,30
First initiator of last session of session set	142,01	3,65
Last non-response of last session of session set	141,54	3,33
Last non-response of last session of session set	141,18	4,35

Table 5.13: Resulting information about the found interval and its session sets

This tables shows that in this case the first timer reset point would have been taken, because the cumulative deviation from the calculated average interval of 149,90 is 2,30 seconds, where for other timer reset points this deviation is larger.

Therefore, session set numbers 2 through 6 are periodic and occur with a measured average interval of 149,90 seconds. Session set number 1 was discarded and is therefore not marked as either periodic or aperiodic. As a result, because all of the 'normal' session sets are periodic, it can be concluded that this session set type is a periodic session set type.

Note that the calculated interval is always an estimation, since the recording unit is likely to be separated from an initiator of a session set type. Hence, the messages that are recorded are always recorded a little later than the time at which they were originally sent. This also means that the messages are affected by a certain network delay that may change with time. Finally, there is also the variable fluctuation in the exact moment at which session sets are started, which has been discussed in a previous section of this chapter. Thus, the interval remains an estimation. In the case of the example, it would have been likely that the initiator party used an interval of 150 seconds.

5.4.2 Example algorithm execution

Although the algorithm description has been accompanied by an example, that particular example was very straightforward, which did not cover the more complicated issues that were discussed earlier in this chapter. Appendix A4 discusses a more complicated example of a session set type, consisting of both periodic and aperiodic session sets.

Chapter 6

Toolset description

The algorithms that have been described have also been implemented into Perl scripts. It is only after such an implementation that the algorithms can be applied on the many available SNMP trace files. The sole purpose of this chapter is to describe for each algorithm how it is implemented and also how the results of the four algorithms are linked to each other. The first section of this chapter shall give a general overview of the implemented scripts. In the subsequent subsections, each of the implemented algorithms are described briefly.

6.1 Toolset overview

All four algorithms have been implemented into a single Perl script respectively [8]. Each of these scripts is considered as a phase within the set of scripts. Furthermore, the input that is given to the first algorithm, or phase, is the result from an already available tool called SNMPDump [1]. The following figure shows the relations between each of the involved scripts, which are shown as squares.

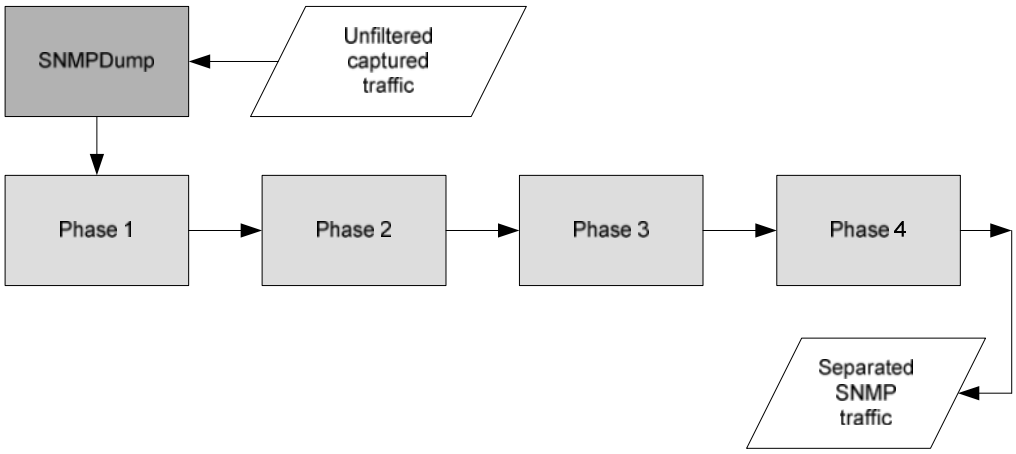

Figure 6.1: General overview of the involved scripts

6.2 Session determination algorithm

The first implemented algorithm, phase 1, is responsible for detecting sessions based on an unfiltered input consisting of SNMP messages. These SNMP messages are given as input via a CSV file and result in multiple smaller CSV files, each containing a small portion of the given input.

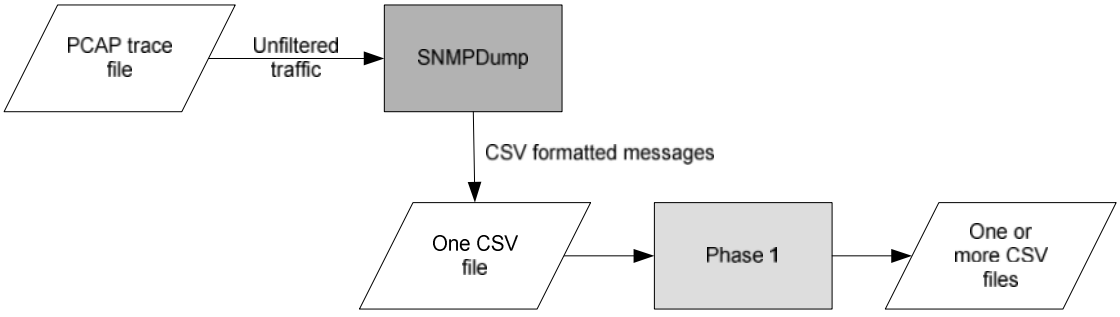

Figure 6.2: Input and output of this first algorithm

Following are the descriptions of the input to this respective script and also the generated output.

6.2.1 Script input

The input of this algorithm is a single CSV file consisting of one or more SNMP messages, where on each line in the given CSV file a single SNMP message is described. It should be noted that these CSV files are created by an already existing tool, called SNMPDump. That particular tool distils all SNMP messages from a PCAP trace file and writes the found SNMP messages to a single CSV file.

Besides the SNMP messages, it is also possible to specify a number of parameters with the start of this script. The following table contains the parameters that can be specified:

Parameter	Purpose
-p [seconds]	Maximum processing time
-r [seconds]	Maximum timeout

Table 6.1: Most significant input parameters for the first implemented script

The maximum processing time parameters allows the user to specify the largest amount of time between the last response and a subsequent request message within a session. This applies to table retrieval process scenarios in which, for example, a manager has received responses to all outstanding requests and it than needs to send a new request, as part of the table retrieval process. If the time between this particular request and the last listed response for a particular session exceeds the limit, then that session shall be considered as ended and the next non-response messages shall be assigned to different or new sessions. As a result, this value should not be set too restrictive.

The second parameter specifies the maximum time between the original message and a retransmission, in order for the last one to be considered as such. Also this value should not be set too restrictive, in order to avoid scenarios in which a new session is incorrectly considered to have started, while in reality the involved message is just a retransmission of an already listed message in a particular session.

6.2.2 Script output

As has already briefly been mentioned, the output of this phase script consists of one or more CSV files. A different CSV file shall be created for each found session. All of these output CSV files are put into a single output folder, each containing one or more lines. Each line represents a single SNMP message that is assigned to that respective session.

Each of the output CSV files has a name that is of a specific format, which is shown here:

SESSION_OPERATION_TYPE-INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-DESTINATIONIP_ADDRESS[DESTINATION_PORT]-[SESSIONNR].CSV

The session operation type that is mentioned in this file format is, for example, get, get-next or trap. The session number allows enumeration of sessions that have all other fields in common. So, these sessions are of the same operation type, are initiated by the same initiator, with the same port number and also the other party information is the same.

6.3 Session type determination algorithm

The second implemented algorithm, phase 2, is responsible for detecting session types based on an input consisting of sessions. These sessions are given as input via a CSV file respectively and result in one or more CSV files, each containing references to sessions that are of the same session type. Besides that, there are also two other files generated.

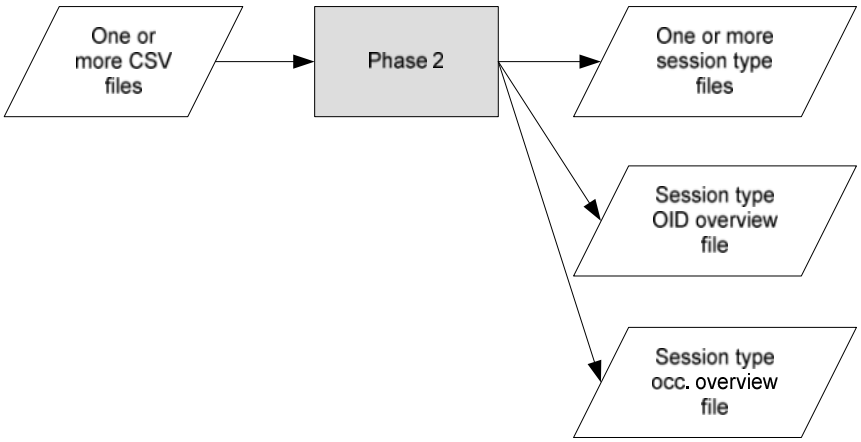

Figure 6.3: Input and output of this second algorithm

Following are the descriptions of the input to this respective script and also the generated output.

6.3.1 Script input

The input of this algorithm is a reference to a folder, which contains of one or more CSV files, each consisting of SNMP messages that are part of the same session.

Besides the sessions, it is also possible to specify a number of parameters with the start of this script. The following table contains the most significant parameter that can be specified:

Parameter	Purpose
-d [0 or 1]	Disregard (initiator) port number

Table 6.2: Most significant input parameters for the second implemented script

This single parameter allows the user to run this script in which it will not consider the initiator port number as part of the session type information. This applies to the operation types get, get-next, get-bulk and set. If this parameter is set to 1 and the script encounters a session with an operation like trap, inform or report, then it will disregard both the initiator port number, as well as the port number of the other party in the session type determination process.

6.3.2 Script output

The output of this phase script consists of one or more CSV files. Each CSV files contains references to sessions (on each line a single reference) that were given as input and that have been marked with the same session type. Therefore, a new CSV file shall be created for each found session type in the output folder of this script.

Each of the output CSV files has a name that is of a specific format, which is shown here:

OPERATION_TYPE-INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-DESTINATIONIP_ADDRESS[DESTINATION_PORT]-[SESSION_TYPE_NR].CSV

The operation type that is mentioned in this file format is, for example, get, get-next or trap. The session number allows enumeration of session types that have all other fields in common. So, these respective sessions are initiated by the same initiator, with the same port number and where also the other party information is the same. Also, the session operation type must be the same. In case the parameter is set, which has been discussed in the previous subsection, then the port number fields may be left blank in selected situations.

Besides these CSV files, there are also two other files generated. One is the session type OID overview file. This file contains for each session type the set of found OID prefixes that are determined to be specific for a particular session type. This information in this file can be found for every found session type, where each line contains the found OID prefixes per found session type.

The other file (the session type occurrence overview file) contains a chronological overview of the session type occurrences, based on the sessions that have been given as input to this algorithm. Each line in this file contains a reference to a session and the specific session type it is of. This chronological overview allows for more easy and optimized processing in the subsequent scripts, which in turn do not need to load all session and session type information.

6.4 Session set and session set type determination algorithm

The third implemented algorithm, phase 3, is responsible for detecting session set and their respective session set type based on an input consisting of sessions and their session types. These are given to this algorithm by means of a chronological overview of the session type occurrences. As output are CSV files, one for each found session set. Moreover, there is for each session set type a file which describes the session sets that are of that particular session set type.

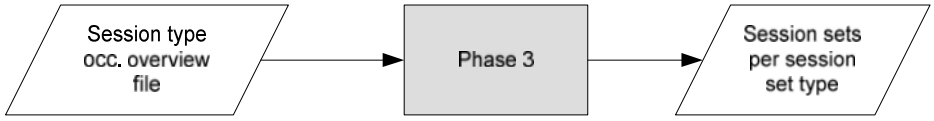

Figure 6.4: Input and output of this third algorithm

Following are the descriptions of the input to this respective script and also the generated output.

6.4.1 Script input

The input of this algorithm consists of a reference to the folder containing all the files that have been generated by the second algorithm. This third algorithm will only use the chronological overview file, which contains all the information needed for this script. It is therefore not required for this algorithm to load all sessions and session type information again. An abstraction, as given with this general overview file, is enough to determine to which session set a particular session shall be assigned.

Besides this general overview of session type occurrences, it is also possible to specify a number of parameters with the start of this script. The following table contains the most significant parameter that can be specified:

Parameter	**Purpose**
-s [0 or 1]	General information overview only
-c [number]	Consider only first *c* hours of the input
-d [0 or 1]	Allow duplicates of session types within a single session set
-e [number]	MINIMUM_SUPER_TYPE_INSTANCE_OCCURENCE_MULTIPLIER
-f [number]	MINIMUM_SUPER_TYPE_INSTANCE_SMALLEST_OCCURENCE
-g [number]	FLOAT_ASSIGN_INTERVAL_DEVIATION_FACTOR
-k [number]	FLOAT_ASSIGN_MAX_INSTANCE_LENGTH_DIFFERENCE
-u [number]	FLOAT_ASSIGN_MAX_INSTANCE_LENGTH_DIFFERENCE_MERGE
-p [number]	FLOAT_ASSIGN_MIN_TYPE_MATCH_FACTOR
-t [number]	FLOAT_ASSIGN_MIN_TYPE_MATCH_FACTOR_START_END
-z [number]	FLOAT_ASSIGN_MIN_TYPE_MATCH_MERGE_FACTOR
-q [seconds]	MINIMUM_TIMEOUT
-b [seconds]	MAXIMUM_TIMEOUT
-r [seconds]	DISCARD_BORDER_INSTANCES_SEEK_RANGE
-y [seconds]	FORCE_TIME_OUT

Table 6.3: Most significant input parameters for the third implemented script

The first parameter allows the user to run the script in a specific mode in which only general information about the involved initiators shall be given. This information includes the number of sessions initiated by that particular initiator, but also an estimated timeout value that can be used for that initiator. This value will be chosen for that particular initiator and is automatically calculated, but can also be overwritten by setting the –y parameter.

The second parameter allows the user to let the algorithm consider only the sessions that occur within the first specifiable number of hours. This allows the user to get a quick result, without having to consider all sessions, which may take a very long time in some cases.

The remaining parameters can be used to adjust very specific aspects of the algorithm. It is not required to specify any of these values, but the user may choose to do so in the case of some specific trace conditions, which may desire such changes. The reader is advised to read the extensive descriptions of these parameters given in the code of this script.

6.4.2 Script output

The output of this phase script consists of two types of output CSV files that are stored in a single result folder. One type encompasses CSV files that contain information about a specific session set. The contents of such a file describes the assigned session type occurrences, the start time of each assigned session, the last request time and also the last response time. The second type of CSV files describes in summary the session sets that are considered to be of the same session set type. Such a file exists for each session set type and lists the following for each session set that is of that session set type: the number of assigned sessions, the start time of the first session, the start time of the last session, the time of the last request in the last session and the time of the last response in the last session.

Each of the output CSV files has a name that is of a specific format. A session set file has the following format:

INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-[SESSION_SET_TYPE_NR]-[SESSION_SET_TYPE_OCCURRENCE_NR].CSV

The contents of the initiator port number field may be left empty, depending on the preferences chosen in phase 2. Besides that field, the session set type number allows for differentiation between different session set types that encompasses session sets initiated by the same initiator. The last field, the session set type occurrence number, is an integer value that indicates this session set number, which is considered to be of a particular session set type.

The remaining result type file, a session set type file, is of the following format:

INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-[SESSION_SET_TYPE_NR].CSV

As can be seen, this is the same as in the last case, but with the exception that the session set type occurrence number field is not included. Such a file will be generated for every session set type that is defined.

6.5 Periodic/aperiodic separation algorithm

The fourth and last implemented algorithm, phase 4, is responsible for determining which session sets, belonging to a particular session set type, are aperiodic and which are to be considered aperiodic. These session sets are given to this algorithm by giving the contents of the output folder of the previous algorithm as input to this script. The output of this script encompasses per session set type all the session sets that are considered aperiodic, or are contributing to a specific interval and are thus considered periodic, or are discarded.

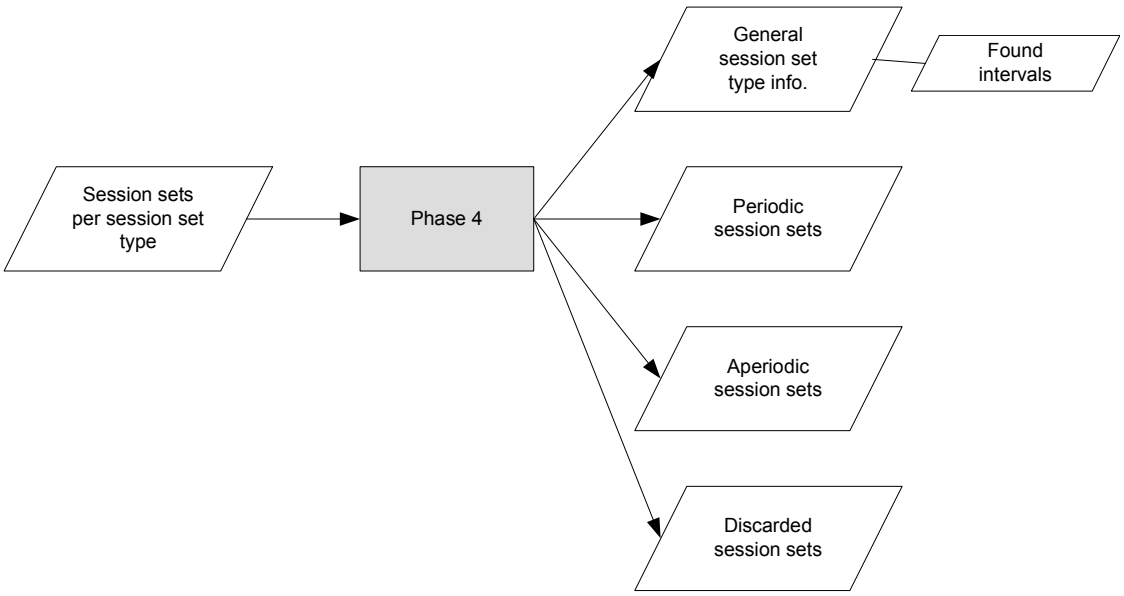

Figure 6.5: Input and output of this fourth algorithm

Following are the descriptions of the input to this respective script and also the generated output.

6.5.1 Script input

The input of this algorithm consists of a reference to the folder containing all the files that have been generated by the third algorithm. This fourth and last algorithm will load the session set information per session set type. This avoids considering session sets that are of different session set types.

Besides specifying the path of the folder containing the information of session sets per session set type, it is also possible to specify a number of parameters with the start of this script. The following table contains the most significant parameter that can be specified:

Parameter	Purpose
-b [number]	FLOAT_ASSIGN_MAX_INSTANCE_LENGTH_DIFFERENCE
-c [seconds]	MAX_INTERVAL_DEVIATION
-d [seconds]	MAX_AVERAGE_INTERVAL_DEVIATION
-e [number]	MIN_INSTANCES_IN_INTERVAL_SEGMENT
-f [seconds]	MAX_TIME_BETWEEN_INSTANCES
-g [seconds]	MAX_SEEK_RANGE

Table 6.4: Most significant input parameters for the fourth implemented script

These parameters can be used to adjust very specific aspects of this algorithm. It is not required to specify any of these values, but the user may choose to do so in the case of some specific trace conditions which may desire such changes. The reader is advised to read the extensive descriptions of these parameters given in the code of this script.

6.5.2 Script output

The output of this phase script consists of two types of output CSV files. One type involves result files that describe the overall result for a specific session set type. Such a file contains information about whether or not the first session set of that session set type has been discarded, because it was considered incomplete. It also contains information about every found interval within the session sets of this session set type. In case

there are session sets considered to be aperiodic, then this file will also contain information about this fact and the number of session sets that are marked as such.

Output files that are of this first type, which are generated for every session set type given as input to this algorithm, are of the following format:

INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-[SESSION_SET_TYPE_NR].CSV

It should be noted that the contents of the initiator port number field may be left empty, depending on the preferences chosen in phase 2.

Besides this type of output CSV file, there is also another type, which contains more specific information. These files contain information about the session sets that are contributing to a single interval within a session set type, or contain information on the session sets that are marked aperiodic, or have been discarded due to expected incompleteness.

The files that contain such specific information are of the following general format:

INITIATORIP_ADDRESS[INITIATOR_PORT_NUMBER]-[SESSION_SET_TYPE_NR]-[MARKER].CSV

This format is practically the same as the previous one, but with the addition of the marker field. This field signifies whether this file contains information about session sets that are contributing to a single interval, in which case it will contain an integer, or it may be the character ‘A’, if it contains information about the aperiodically marked session sets. In case a session set is considered to be incomplete, then it will be referenced in a file of this type with a ‘D’ as marker.

Chapter 7

Analysis results

The determination of the four algorithms and their implementation resulted in a toolset consisting of four phase scripts, as described in the previous chapter. After this, it is possible to give some of the available SNMP trace files as input to this set of scripts. This has been done with four trace files, where each has been generated on a different network location. This chapter describes the results that have been found for each trace file respectively. Due to applicable privacy regulations and information release policies, no location or content specific information shall be described.

7.1 Trace I01t01

This trace is recorded on a national research network, which contains 24 hours of recorded SNMP traffic. The following tables lists the results from this trace that have been generated, after giving this trace file as input to the set of four phase scripts.

Description	**Value**
Considered part of the trace file	100%
SNMP messages processed	71501
Unhandled SNMP messages	544
Found retransmissions	9,9% of all SNMP messages
Sessions found	• 5321 get-next sessions • 2636 get sessions • 127 set sessions • 8084 in total
Script parameters	• Max. time between retransmission: 25 seconds • Max. processing time: 4 seconds
Script processing time	~90 minutes

Table 7.1: Information about phase 1 results

This table lists the results after giving the single CSV file, representing this trace file, as input to the first phase script. It shows that there were 544 SNMP messages that could not be assigned to any session. These are all response messages. This could be the result of the fact that some requests have not been recorded and are therefore not included into the trace file. As a result, the respective response message cannot be assigned to sessions, because there is no existing session with the respective request listed.

The found number of retransmissions seems quite high in comparison to the other considered trace files. This is likely the result of the fact that a lot of the listed requests are accompanied by two immediate retransmissions that all occur within a second from the first request being sent.

Another noteworthy aspect is the relatively long script processing time. This is because of the fact that the phase script used to process this trace file was one of the earlier versions, which was not yet optimized.

In the algorithm description of this respective phase, a number of considerations have been made, which have been described in chapter 2.2. One of the considerations was about holes in tables. Inspection of this trace file suggests that there are at least two tables being referenced which contain discontinuities (i.e. the last part of the OID of the referenced column items need not be equal to the previously referenced OID for that column plus one). This has been detected for tables like the cardIfIndex table (1.3.6.1.4.1.9.3.6.13), where a device cannot be found for every index. The same can be said about the detected entPhysicalTable (1.3.6.1.2.1.47.1.1.1). But, because the manager in question uses get-next requests to retrieve the contents of these tables, no irregularity and therefore no discontinuity between the response-non-request relations has been found.

Besides this consideration, also no unexpected OID insertions have been detected in this trace file, or a change in the order of column references during the retrieval of the contents of a table or column.

Description	Value
Session types found	• 3726 get-next session types • 1650 get session types • 127 set session types • 5503 different session types in total
Script parameters	• Disregard port number of initiator in session type discovery
Processing time	10 minutes

Table 7.2: Information about phase 2 results

This table summarizes the results from the phase 2 script, after giving the results from the phase 1 script as input to it. An important aspect of this result is the relatively large number of session types found with respect to the number of found sessions. This suggests that there are a very large number of unequal sessions in this trace file.

Description	Value
Session sets found	39
Session set types found	29
Script parameters	• No timeout enforcement (auto detect was used) • Multiple occurrences of a session type within a session set were NOT accepted
Processing time	1 day, 23 hours and 45 minutes

Table 7.3: Information about phase 3 results

This table shows the results from the phase 3 script execution on this trace. The results show that there are very few session sets considered to be of the same session set type. This already suggests that there is a very large percentage of aperiodic behaviour in this trace file.

Description	Value
Periodic session sets	0
Aperiodic session sets	39 (100%)
Script parameters	Default settings
Processing time	< 1 minute

Table 7.4: Information about phase 4 results

The results of this last script show that no session sets have been marked as periodic, so no intervals have been found. All session sets are considered aperiodic, because only one session set type describes three session sets, which do not have an identifiable interval. All other session set types describe only two or just one session set, for which automatically aperiodicity is concluded. A different report [9] on this particular trace confirms this finding.

In chapter 5.2, a number of considerations have been made. In this trace were no interruptions in periodic behaviour (e.g. interval changes) detected. Also, there have no session set types been found which involve session sets that form a composite of both periodic and aperiodic behaviour.

7.2 Trace l03t02

This trace is recorded on a university network, which contains 6,5 days of recorded SNMP traffic. The following tables lists the results from this trace that have been generated after giving this trace file as input to the set of four phase scripts.

Description	Value
Considered part of the trace file	First hour
SNMP messages processed	361000
Unhandled SNMP messages	0
Found retransmissions	80469
Sessions found	• 5157 get-next sessions • 1783 get sessions • 20940 in total
Script parameters	• Max. time between retransmission: 25 seconds • Max. processing time: 4 seconds
Script processing time	15 minutes

Table 7.5: Information about phase 1 results

This table lists the results after giving the single CSV file, representing this trace file, as input to the first phase script. The script execution went relatively fast, because the phase script used to process this trace file was more optimized than in the case of the previous trace.

Inspection of this trace file suggests that there were no unexpected OID insertions, or a change in the order of OIDs during the retrieval of the contents of a table or column have been detected.

Another result of this inspection is that there is at least one table being referenced which contains discontinuities. This has been detected in the ifTable, where not for every IP address a listing can be found. But, because the manager in question uses get-next requests to retrieve the contents of these tables, no irregularity and therefore no discontinuity as such between the response-non-request relations has been found.

Description	Value
Session types found	• 57 get-next session types • 1220 get session types • 1277 in total
Script parameters	• Disregard port number of initiator in session type discovery
Processing time	8 minutes

Table 7.6: Information about phase 2 results

This table summarizes the results from the phase 2 script, after giving the results from the phase 1 script as input to it. An important aspect of this result is the small number of session types found with respect to the number of sessions. This suggests that there are a very large number of sessions in this trace file that are of the same session type.

Description	Value
Session sets found	4734
Session set types found	43
Script parameters	• No timeout enforcement (auto detect was used) • Multiple occurrences of a session type within a session set were NOT accepted
Processing time	~ 6 days

Table 7.7: Information about phase 3 results

This table shows the results from the phase 3 script execution on this trace. The results show that there are on average multiple session sets considered to be of the same session set type. Another interesting aspect of this considered trace segment is that there are seven unique initiators found. One of these initiators is responsible for 35 of all session set types found. However, only two of these initiators were responsible for creating session set types with more than two session sets.

Description	Value
Periodic session sets	12, involving 14508 sessions (interval 900 seconds) 20, involving 40 sessions (interval 180 seconds) 34, involving 492 sessions (interval 300 seconds) 6, involving 12 sessions (interval ~55 seconds)
Aperiodic session sets	4662
Script parameters	• MAX_INTERVAL_DEVIATION=10 • MAX_AVERAGE_INTERVAL_DEVIATION=2
Processing time	< 1 minute

Table 7.8: Information about phase 4 results

The results of this last script show that in this trace both periodic and aperiodic session sets have been found. Although by far the greatest portion of all sessions is assigned to session sets that are periodic, there are still a large number of session sets that are considered to be aperiodic. In the case of this trace, there is one particular reason for this. Many sessions that are of a particular session type occur in some polling instances more than once. Because the phase 3 script was executed with the parameter set to not accept multiple occurrences of the same session type within a single session set, this resulted in many session set types which just describe one or more session sets that only contain a single session each. As a result, all of the respective session sets are considered aperiodic, because no interval can be determined for these irregularly occurring sessions.

In chapter 5.2 have a number of considerations been made. In this trace no interruptions in periodic behaviour were detected. However, there is one session set type which was found and which describes session sets that form a composite of both periodic and aperiodic behaviour. This occurs in the case of a specific session set type that only encompasses session sets that contain trap sessions.

7.3 Trace l04t01

This trace is recorded at a server-hosting provider, which contains 4 hours of recorded SNMP traffic. The following tables list the results from this trace that have been generated after giving this trace file as input to the set of four phase scripts.

Description	Value
Considered part of the trace file	100%
Unhandled SNMP messages	0
SNMP messages processed	15099
Found retransmissions	431 (2,9%)
Sessions found	• 192 get-next sessions • 2785 get-bulk sessions • 4527 get sessions • 7504 sessions in total
Script parameters	Only default settings used
Script processing time	< 1 minute

Table 7.9: Information about phase 1 results

This table lists the results after giving the single CSV file, representing this trace file, as input to the first phase script. The script execution went relatively fast, because the phase script used to process this trace file was more optimized than in the case of the first trace. Another factor is that the sessions detected in this trace seem to be spaced apart further than in the other two traces discussed so far.

Inspection of this trace file suggests that there were no tables referenced in this trace which contain discontinuities. Besides this consideration, also no unexpected OID insertions, or a change in the order of OIDs during the retrieval of the contents of a table or column have been detected.

Description	**Value**
Session types found	• 58 get-bulk • 4 get-next • 95 get session types • 157 session types in total
Script parameters	• Disregard port number of initiator in session type discovery
Processing time	< 1 minute

Table 7.10: Information about phase 2 results

This table summarizes the results from the phase 2 script, after giving the results from the phase 1 script as input to it. An important aspect of this result is the small number of session types found with respect to the number of sessions. This suggests that there are a very large number of sessions in this trace file that are of the same session type.

Description	**Value**
Session sets found	737
Session set types found	16
Script parameters	• No timeout enforcement (auto detect was used) • Multiple occurrences of a session type within a session set were NOT accepted
Processing time	49 minutes

Table 7.11: Information about phase 3 results

This table shows that 16 session set types were found, while in reality there may have been fewer session set types. This is likely the result of a too restrictive timeout used between sessions that make up a session set. Manual inspection of this trace file suggests that there are very large gaps between some sessions initiated by one particular IP address, which may in reality still be part of the same polling instance. In some cases, this gap is larger than 60 seconds. Besides load spreading by an intelligent manager application, this phenomenon could also be the result of the possibility that some traffic, involving certain polling instances, has never been seen by the trace file recording unit. Nevertheless, at this moment it is considered not to be realistic to suggest that sessions occurring more than 60 seconds after its predecessor are still part of the same session set.

Another import observation regarding this trace file is that information about the involved network elements states that the primary manager of that network operated on a system with three different IP addresses. The number of sessions per initiator that were detected in this trace confirm this. It should be added that even after considering the three initiators to be the same, there would still be very large gaps between some sessions that may in reality belong to a single polling instance.

It should also be noted that there were some irregularly occurring session types found. This is likely the result of the fact that some agents did not respond in some polling instances, where in other polling instances they did react.

Description	Value
Periodic session sets	147, involving 6785 sessions (interval 300 seconds)
Ambiguous session sets	590, involving 719 sessions
Script parameters	Default settings
Processing time	< 1 minute

Table 7.12: Information about phase 4 results

The results of this last script execution show that one session set type, involving 49 session sets, initiated by a particular IP address, are all considered to be periodic. An interval of 300 seconds was detected, no session sets of this session set type were considered aperiodic. The remaining session set types were initiated by either of the other two initiators. The respective session sets involve those that may have been part of just one session set type, as a result of the described anomaly involving this initiator. But, because the cause of this phenomenon remains questionable, some session sets of these session set types have been marked as ambiguous, instead of either periodic or aperiodic. In total 92% of all sessions are assigned to session sets that are marked periodic and 8% are marked ambiguous. If the cause of this phenomenon can be determined, then it may be that 100%, or almost that part, of all sessions can surely be considered part of session sets that are marked periodic.

In chapter 5.2, a number of considerations have been made. In this trace no interruptions in periodic behaviour were detected, nor were there session sets detected that belong a session set type which form a composite of both periodic and aperiodic behaviour.

7.4 Trace l05t01

This trace is recorded on a German governmental network, which contains 24 days of recorded SNMP traffic. The following tables lists the results from this trace that have been generated after giving this trace file as input to the set of four phase scripts.

Description	Value
Considered part of the trace file	First 24 hours
SNMP messages processed	1121000
Unhandled SNMP messages	0
Found retransmissions	40
Sessions found	• 6 get-next sessions • 573395 get sessions • 4 trap2 sessions • 573405 sessions in total
Script parameters	• Max time between retransmission: 25 sec. • Max processing time: 4 seconds
Script processing time	9 minutes

Table 7.13: Information about phase 1 results

This table lists the results after giving the single CSV file, representing this trace file, as input to the first phase script. The script execution went relatively fast, because the phase script used to process this trace file was more optimized than in the case of the first trace.

Inspection of this trace file suggests that there were no unexpected OID insertions, or a change in the order of OIDs during the retrieval of the contents of a table or column have been detected.

Description	Value
Session types found	• 4 get-next session types • 129 get session types • 3 trap2 session types • 136 session types in total
Script parameters	• Disregard port number of initiator in session type discovery
Processing time	1 hour and 45 minutes

Table 7.14: Information about phase 2 results

This table summarizes the results from the phase 2 script, after giving the results from the phase 1 script as input to it. An important aspect of this result is the small number of session types found with respect to the number of sessions. This suggests that there are a very large number of sessions in this trace file that are of the same session type.

Description	Value
Session sets found	12381 or 4239
Session set types found	15 or 8
Script parameters	• No timeout enforcement (auto detect was used) • Multiple occurrences of a session type within a session set were NOT accepted
Processing time	5 days, 2 hours and 4 minutes

Table 7.15: Information about phase 3 results

This script was executed with the parameter set which disallows multiples sessions of the same session type to exist in the same session set. However, manual inspection suggests that sessions of the same session type exist within the same polling instance and should therefore be considered part of the same session set. Merging these session set manually yields 4239 session sets and 8 session set types.

Another observation of the inspection of this trace segment shows that there were no irregularly occurring session types found within a set of session sets that belong to the same session set type.

Description	Value
Periodic session sets	4229 (20 seconds)
Ambiguous session sets	10
Script parameters	Default settings used
Processing time	~ 5 minutes

Table 7.16: Information about phase 4 results

The results of this last script execution shows that the largest session set type, describing 4229 session sets, are all periodic. All of these session sets contribute to a single interval of 20 seconds. The remaining ten session sets were initiated by different network element, forming all session set types which each encompass three session sets or less. No periodicity could be found in any of these session sets, resulting in ten aperiodic session sets.

In chapter 5.2, a number of considerations have been made. In this trace no interruptions in periodic behaviour were detected, nor were there session sets detected that belong to a session set type which form a composite of both periodic and aperiodic behaviour.

Chapter 8

Conclusions

8.1 Research findings

Based on the main research goal of this thesis, as stated in chapter 1, it can be stated that significant progress is made in the field of research regarding the complete separation of periodic and aperiodic SNMP traffic in SNMP traces. Although the devised methods and algorithms theoretically suggest a proper way of achieving this goal, the actual application of the generated scripts on the available SNMP traces suggests that there may still be some research required.

As has been shown in the previous chapter, almost all periodic SNMP traffic could be separated completely and without any uncertainty. However, a small percentage of the SNMP traffic is considered ambiguous and therefore no complete separation of the two types of SNMP traffic could yet be attained for every available SNMP trace file. One of the possible reasons for this may be that absolute and complete separation of the two types of traffic may not be possible in every situation due to the lack of completeness of a trace file. This and other possible reasons for this issue have been discussed in chapter 7.3.

At the beginning of this thesis were the central research questions posed and described briefly. These questions have been answered in the chapters that followed. These questions and their respective answers will be briefly summarized in the remainder of this chapter.

How to determine which SNMP messages belong to a single session?

Chapter 2 discussed the created algorithm for this problem in much detail. The core aspects of this algorithm include the use of a separation method, which filters SNMP messages, such that only SNMP messages belonging to the interaction between two network elements remain. Another filter separates all SNMP messages that are of a certain SNMP operation type (and the possibly related response messages) into their respective categories. A timeout mechanism will then be used to select individual sessions.

The application of this implemented algorithm on a couple of SNMP trace files (chapter 7) suggests that this algorithm was properly implemented and that the consideration of sessions as a means of relating single SNMP messages is a correct approach and proper basis for the larger relationships, like session sets.

Chapter 2.2 discussed a number of possible scenarios that have been taken into account during the creation of this algorithm. After exposing this implemented algorithm to the four real world trace files, it can be concluded that multiple OID references within the same message, retransmissions and sessions other than get sessions have been seen in these trace files. However, unequal column lengths of columns that make up a particular table, have not been seen in any of these trace files. The same can be said about holes in columns or tables which result in a discontinuity in the relations between the non-response and response messages that are part of a single session. This means that discontinuities that can be found in some tables, like the ifTable, do not result in discontinuities of the non-response and response relationships within a single session. This is the result of the fact that managers make use of get-next requests in order to retrieve the contents of these kinds of tables. Moreover, no sessions have been found that contain irregular OID insertions, or have column reference order changes during a table or column retrieval process.

How to determine the session type of a session?

After sessions could be identified, the next step was the identification of session types. Chapter 3 discussed this algorithm thoroughly, which showed a method that would involve the identification of the operation

type of the non-response messages in a session, as well as the OID prefixes of the referenced OIDs. It has been described that for different operation types different methods of identification were used.

The developed algorithm and its implementation seem to be a proper way to differentiate between different sessions and therefore allow for easy comparison of sessions in subsequent algorithms.

How to determine which sessions form session sets?

Being able to identify sessions forms a basis for the possibility of looking for larger relationships. Clearly, as stated in chapter 1.2, sessions hardly occur independent of others. In most cases, there is a clear relationship between the (regular) occurrences of certain session types. In the developed algorithm, as discussed thoroughly in chapter 4, the occurrences of session types during the course of a certain time window are compared for proximity. If two or more session types have a significant number of their occurrences within a certain time range of one another, then these are considered to be related. These related session types, and thereby the occurrences of their respective sessions, form session sets of a specific session set type, which each in turn consist of one or more sessions.

Also for this algorithm a number of possible problematic scenarios have been discussed in chapter 4.2. After applying this implemented algorithm to the available real world trace files, it is shown that multiple initiators on a single network, session set types encompassing session sets of which some are behaving periodically and others aperiodic have all been found in one or more trace files. Besides that, also incomplete session sets and irregularly occurring session types have been found. On the other hand, no scenarios have been found that suggest that there are multiple managers operating from a single IP address. However, there was one trace file that contained SNMP traffic that originated from a single manager that operated from three different IP addresses.

When is a set of sessions considered (a)periodic?

After a session set type has been identified, the next step is the determination of whether this session set type and its session sets are to be considered periodic or aperiodic. Only when enough of the respective session sets occur in a regular fashion, based on one fixed interval, shall this subset of session sets be considered periodic. It shall be marked as aperiodic traffic in most other cases. However, there remains some room for specific traffic, as stated in chapter 7.3, which is to be considered neither of the above. This kind of traffic will be considered as ambiguous, meaning that it could be either of both traffic types. This is the result of the fact that in some scenarios too little is known about the causes of certain SNMP traffic, which as such leaves too much uncertainty on whether or not it is periodic or not. A number of possible reasons and scenarios have been discussed in that particular section.

A number of considerations have been made for the development of this algorithm, as described in chapter 5.2. The real world trace files contain cases in which for a particular session set type, there are some session sets that are marked as periodic and others are marked aperiodic.

How can intervals in SNMP traces be determined?

Intervals can only be found in the periodic SNMP traffic segment. Hence, it is required that a complete set of SNMP traffic first be separated into the two types of traffic by using the toolset which is explained in-dept in chapter 6 and its application in chapter 7. The last script of this toolset also determines all available intervals in every session set type for which at least some of the respective session sets are considered to be periodic. The result of this application contains all found intervals, as well as their level of contribution to the subset of traffic as a whole.

8.2 Recommendations

Although significant research results have been described in this thesis with regards to the complete separation of the two types of SNMP traffic, there still remains some ground uncovered. Therefore, this final chapter will list some suggestions for future research projects that may be considered in order to achieve the absolute and complete separation of these traffic types, where this is possible. The next list also suggests future research topics, based on the findings of this thesis.

8.2.1 Toolset extension suggestions

Some suggestions with regard to the existing set of tools that were either used and/or developed in the process of this thesis assignment are the following:

- The first research suggestion would be about polling consistency. This would require an analysis of the existing programming implementations of manager applications. One might want to look for situations and conditions in which certain SNMP managers will halt the execution of an active polling sequence. The current toolset may need to be adjusted or extended, in order to cover these potentially exceptional situations;
- Another suggestion would involve research on whether or not certain SNMP management software is programmed to spread the network load of its polling activities over a certain amount of time between two start points of polling instances. This could be one of the potential causes of some of the encountered problems with the existing toolset;
- At the moment, the toolset searches for session type occurrences and attempts to assign these to session sets. It would be of much interest to know, if and under which circumstances there are relations between the occurrences of SNMP messages with different operation types and that are initiated by different initiators. For example, is there a relation between the occurrence of a trap message, initiated by an agent, and a possible polling of that agent by the manager that received that trap message?

8.2.2 Future research topics

The following topic has become potentially interesting for future research:

- Now that the separation of the two kinds of traffic is possible, research is required to find a definitive list of causes of aperiodic and periodic traffic. It would also be interesting to understand more about the potential relationship between a certain aperiodic activity and the actual aperiodic SNMP traffic that this causes;

Appendix

A1 Example algorithm execution – session detection

In order to give a better understanding of how all of the first algorithm steps work together - yielding one or more sessions - here follows an example of a situation for the algorithm that is described in chapter 2. This situation is not based on a real world trace, but is created artificially in order to highlight some of the most important aspects of the defined algorithm.

Consider the following set of recorded SNMP messages as input for the algorithm:

Time (s.)	**Source IP**	**Source Port**	**Destination IP**	**Destination Port**	**Operation**	**Request ID**	**OIDs**
0,1	C	161	A	1100	response	4	α
0,35	A	1100	B	161	get-next	10	β, γ
2,11	A	1099	E	161	get-req	9	μ.0
2,20	E	161	A	1099	response	9	μ.0
2,35	A	1100	B	161	get-next	10	β, γ
2,38	B	161	A	1100	response	10	β.0, γ.0
2,45	A	1100	B	161	get-next	12	β.0, γ.0
2,51	B	161	A	1100	response	12	β.1, ε.1
2,52	D	161	F	2290	trap	91	α.0
2,59	B	161	A	1100	response	10	β.0, γ.0
2,62	A	1100	B	161	get-next	15	β.1
2,71	B	161	A	1100	response	15	γ.0
30,35	A	1101	B	161	get-next	16	β, γ

Table A1.1: Approximately 30 seconds of recorded SNMP messages

At the beginning of the algorithm, no open session yet exist. The algorithm, as depicted in figure 2.9, begins at the first point. It loads the first message: a response message. Then it goes to point two, where it is determined that no session is willing to accept this message, simply because there is no open session available. As a result, the next step is point 3, where the response message is discarded. As has already been suggested, at the moment at which the recording of messages starts, the recording may begin recording messages belonging to already active sessions. Therefore, just like in this case, these response messages ought to be discarded, if no session accepts them. Since more messages are available in the input, the algorithm now returns to point 1.

The second message is a get-next request coming from manager *A*, which is sent to agent *B*. Requested are the first values of columns β and γ respectively. Since this is a request message, the algorithm goes from point 1 to point 5 and then to point 6, since still no open sessions exist. But, in contrast to the previous message, this request will cause the creation of a new and open session. This new session will contain only this single get-request at this point. Again, more input is still available, so the algorithm goes back to point 1, considering the next message. The following single open session exists at this point.

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
0,35	A	1100	B	161	get-next	10	β, γ

Table A1.2: Open session #1 (not yet closed, intermediate result)

The third message is a get-request, not a get-next request as the previous one. Since this message takes place between two different network elements, it cannot be part of the existing open session that is bound to a relation between manager *A* and agent *B*. This is just one of the many restrictions that this message does not meet, in order for it to be assigned to the existing open session. As a result, another new open session is created which will contain only this get-request message:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
2,11	A	1099	E	161	get-req	9	μ.0

Table A1.3: Open session #2 (not yet closed, intermediate result)

The fourth message is a response to the third message. Because the source and destination information of this response match the destination and source information respectively of the listed request this response message will be assigned to that second open session. Other reasons for acceptance are the fact that it also matches the request ID of the listed request and the response occurs in a timely fashion after the listed request message:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
2,11	A	1099	E	161	get-req	9	μ.0
2,20	E	161	A	1099	response	9	μ.0

Table A1.4: Open session #2 (not yet closed, intermediate result)

The fifth message is clearly a retransmission of message 2. And because it matches the criteria to be considered as a retransmission, it will be assigned to the first open session.

The sixth message is either a delayed response to the request with request ID 10, or a response to the retransmission of that request with the same request ID. In either case, it matches the criteria for this message to be assigned to the first open session. This open session will by now contain the following messages:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
0,35	A	1100	B	161	get-next	10	β, γ
2,35	A	1100	B	161	get-next	10	β, γ
2,38	B	161	A	1100	response	10	β.0, γ.0

Table A1.5: Open session #1 (not yet closed, intermediate result)

The seventh message shows a request message, which seems to be a continuation of a table retrieval process, based on the last received response by the manager. Since both OIDs in this request are equal to both OIDs in the last listed response of open session 1, this message can be considered a continuation of open session 1. Therefore, it is added to open session 1.

The next message is a response message to this last request. Note that the end of the γ-column has been reached. Because it can be identified as a proper response to a listed request in open session 1, it will be added to that open session:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
0,35	A	1100	B	161	get-next	10	β, γ
2,35	A	1100	B	161	get-next	10	β, γ
2,38	B	161	A	1100	response	10	β.0, γ.0
2,45	A	1100	B	161	get-next	12	β.0, γ.0
2,51	B	161	A	1100	response	12	β.1, ε.1

Table A1.6: Open session #1 (not yet closed, intermediate result)

The ninth message is a trap message that is communicated between an agent and a manager of which no open sessions exist yet. More generally, there is no open session for trap messages that would be willing to accept this message. Therefore, a third open session is created:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
2,52	D	161	F	2290	trap	91	α.0

Table A1.7: Open session #3 (not yet closed, intermediate result)

The tenth, eleventh and twelfth recorded message all seem to belong to the first open session. The tenth message is no doubt a proper member of open session 1. The eleventh message is a new request message. As stated in the restrictions for requests to be assigned to an existing session, at least one of the OIDs in the request must match one of the OIDs in the last response to the last request. This example shows the importance of not just looking for, and comparing with the last chronologically listed response message in an open session. If that would not have been done, this request (the eleventh message) would incorrectly be seen as the start of a new open session. Instead, now it is seen as a continuation of the table retrieval process. The twelfth message is simply a response to this request. This yields the following list of member messages, making up open session 1:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
0,35	A	1100	B	161	get-next	10	β, γ
2,35	A	1100	B	161	get-next	10	β, γ
2,38	B	161	A	1100	response	10	β.0, γ.0
2,45	A	1100	B	161	get-next	12	β.0, γ.0
2,51	B	161	A	1100	response	12	β.1, ε.1
2,59	B	161	A	1100	response	10	β.0, γ.0
2,62	A	1100	B	161	get-next	15	β.1
2,71	B	161	A	1100	response	15	γ.0

Table A1.8: Session #1 (session closed, final result)

The last recorded message occurs almost 30 seconds after the last processed message and therefore it cannot be considered a timely resumption of any of the open sessions. Therefore, before this last message is to be considered, all open sessions that have not received any new messages for a certain timeout period,

should be closed and stored as results of this algorithm. As a result, the processing of the last listed message of table A1.1 shall result in a new open session. After that, only one open session will exist.

Since no new messages can be found in the input, the algorithm will go to step 7, where it will close all open session, in this case only one, and store all of them as results of this algorithm execution. In the end, the following sessions will have been detected, besides the listed closed session in table A1.8:

Time (s.)	**Source IP**	**Source Port**	**Destination IP**	**Destination Port**	**Operation**	**Request ID**	**OIDs**
2,11	A	1099	E	161	get-req	9	μ.0
2,20	E	161	A	1099	response	9	μ.0

Time (s.)	**Source IP**	**Source Port**	**Destination IP**	**Destination Port**	**Operation**	**Request ID**	**OIDs**
2,52	D	161	F	2290	trap	91	α.0

Time (s.)	**Source IP**	**Source Port**	**Destination IP**	**Destination Port**	**Operation**	**Request ID**	**OIDs**
30,35	A	1101	B	161	get-next	16	β, γ

Table A1.9: Result of sessions 2, 3 and 4 (all sessions closed, final result)

This example algorithm execution has shown the most characteristic elements of the defined algorithm in connection with the related complete definition of the term session.

A2 Example algorithm execution – session type detection

In order to give a better understanding of how all of these steps work together - yielding the respective session types for each session given as input - here follows an example situation for this algorithm. This situation is not based on a real world trace, but is created artificially, in order to highlight some of the most important aspects of the defined algorithm.

Consider the following set of sessions as input for the algorithm that detects session types:

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
0,60	C	161	A	6001	trap	1000	α.0, α.1, β.1
0,61	C	161	A	6001	trap	1000	α.0, α.1, β.1

Table A2.1: Session #1 describing a trap session

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
10,81	A	2800	C	161	get-req	5	β.1, γ.1
10,83	C	161	A	2800	response	5	β.1, γ.1

Table A2.2: Session #2 describing a get session

Time (s.)	Source IP	Source Port	Destination IP	Destination Port	Operation	Request ID	OIDs
44,25	B	4905	D	161	get-next	1	α, β, ε
44,27	D	161	B	4905	response	1	α.0, β.0, ε.0
44,28	B	4905	D	161	get-next	2	α.0, β.0, ε
46,30	B	4905	D	161	get-next	2	α.0, β.0, ε
46,33	D	161	B	4905	response	2	α.1, β.1, ε.0
46,35	B	4905	D	161	get-next	3	α.1, β.1, ε
46,39	D	161	B	4905	response	3	β.1, γ.0, ε.0

Table A2.3: Session #3 describing a get-next session

These three sessions are given as input to the algorithm. Following is a step-by-step description of this algorithm handling the first session:

- The first algorithm is loaded first, as described in the step description in chapter 3. In step 1, the determination is made that it involves a trap session. Then, since there are two messages available, the algorithm goes to step 2;
- Here it loads the first message, the first trap message. Since this is not a response, the algorithm goes to step 3, where the OID prefixes are determined;
- Here follows a description of the sub steps taken in point 3:
    - At this point the list of OID prefixes is still empty. Also, there can no last response be determined, since there is no response message loaded in this session yet. Therefore, all three OIDs will be taken to the next sub step;
    - Now, the first OID of this message will be tested. If it contains a prefix that matches any of the listed OIDs in the list of OID prefixes, then this OID will not be considered further. However, if it does not, then its prefix will be taken and will subsequently be added to the list of OID prefixes. Since this list is currently empty, its prefix (α) will be added to the list of OID prefixes;
    - The second OID shall then be considered. This OID contains a prefix that is equal to the one currently listed in the list of OID prefixes. Hence, it shall not be considered further;
    - The third and last OID is of a different column than the previous two, which results in the addition of the prefix of this OID to the list of OID prefixes. In the end, for these three OIDs α and β shall be taken to the next sub step.

  Now that this message has been dealt with, the algorithm shall return to point 2, where the next message is loaded;
- The next message of this session is a non-response message, thus the algorithm will go to step 3;
- Here follows a description of the sub steps taken in point 3 for this second message:
    - Again, no last response can be found and the only listed elements in the list of OID prefixes are that of the α-column and the β-column;
    - Using the same methodology, now the OIDs of this message shall be checked for matching prefixes. Since all three OIDs are the same as those in the first trap message, it follows that all of these OIDs match one of the prefixes already listed in the list of OID prefixes. As a result, no new OIDs will be added to that list.

  Now that this message has been dealt with, the next message can be processed;
- Since there are no additional messages to be found in this session, the next session shall be loaded and considered in point 1. The result for this first session is the following information, that together encompasses the information of the session type of this session:

Session Type	
Operation Type	Trap
Initiator Party IP Address	C
Initiator Party Port Number	<NOT CONSIDERED>
Other Party IP Address	A
Other Party Port Number	<NOT CONSIDERED>
{OID prefix(es)}	{α, β}

Table A2.4: The session type information for session #1

Note that the port numbers have not been stated, as a result of the algorithm description discussed earlier. Also note that the second message - a retransmission of the first message of this session – did not influence the list of OID prefixes for this session type. A third point of interest is the list of OID prefixes. As is shown, only the columns containing the items referred to in the two trap messages form the list of OID prefixes.

Following is a step-by-step description of this algorithm handling the second session:

- From point 1, the algorithm goes to point 2, after resetting the session type information fields for this session;
- The first message being loaded is a request message and will be handled in point 3;
    - There is still no response loaded yet, so no OID comparison can take place between this request message and the most recent response message;

    - The first OID of this message does not match any of the listed OIDs, since the list of OID prefixes is still empty at this point. The prefix of this OID shall now be extracted, before it will be added to the list of OID prefixes;
    - The second OID refers to a different column and has therefore no matching prefix with the listed OID in the list of OID prefixes. As a result, it shall be added to this list.

  Now that this message has been dealt with, the next message shall be processed;
- The algorithm will return to point 2, since still more messages are available in this session;
- In point 2 the next message is being loaded, but this is a response message, which is not considered in this algorithm. After this message, no other messages can be found and the algorithm returns to point 1; the result for this session is the following information, that in all encompasses the information of the session type of this session:

Session Type	
Operation Type	Get-request
Initiator IP Address	A
Initiator Port Number	<NOT CONSIDERED>
Other Party IP Address	C
Other Party Port Number	161
{OID prefix(es)}	{β.1, γ.1}

Table A2.5: The session type information for session #2

In this case, the initiator of the session was manager *A*, which contacted agent *C*. Also, here is the port number of only one side stated, because of the operation type of this session, which has been explained previously. Finally, the two messages of this session resulted in a list of two OID prefixes.

Following is a step-by-step description of this algorithm handling the third and final session of the input given to this algorithm:

- From point 1, the algorithm goes to point 2 after resetting the session type information fields for this session;
- In point 2, it loads the first message, which is a request. Then, the algorithm goes to step 3 to deal with the OIDs of this request message;
- In point 3, the OIDs of this request shall be compared with those of the last response. But, this is the first message of this session, so no response is loaded yet.
    - Since the list of OID prefixes is empty and the OIDs listed in this first request are not references to the same column / scalar as in a previous response, the table of OID prefixes shall be filled with these three OIDs.
- Then the next message is considered, which is a response, which will not be considered. The third message is again a non-response and will be checked in point 3 of the algorithm overview.
    - Now, a previous response can be found. And this non-response message contains three OIDs out of which two (α.0 and β.0) are equal to one of the listed OIDs of this last response respectively. The third OID in this request (ε) cannot be found in the last response, so it may be a new column or scalar reference. But, because it has a prefix equal to one of the already listed OIDs in the list of OID prefixes, also this third OID shall have no influence on this list. In all, this means that no new columns or specific scalars are being referenced with this new request message. Therefore, all three OIDs in this message shall not be eligible for addition to the list of OID prefixes, which in turn will stay the same.
- The third listed non-response message is a retransmission of the second request message. Since also for this request message the same applies to all of its listed OIDs, as in the second request message, this request shall not have any influence on the listed OIDs in the list of OID prefixes;
- The fourth request message contains in total three OIDs, out of which again two OIDs (α.1, β.1) match exactly those of the response occurring just before it. The third listed OID (ε) is not listed in this last response, but its prefix matches an already listed OID in the list of OID prefixes. Therefore, all three OIDs in this message shall not be eligible for addition to the list of referenced OIDs, which in turn will stay the same;

- After the last response message has been read by this algorithm, the following information shall form the session type information of the found session type:

Session Type	
Operation Type	Get-next-request
Initiator IP Address	B
Initiator Port Number	<NOT CONSIDERED>
Other Party IP Address	D
Other Party Port Number	161
{OID prefix(es)}	{α, β, ε}

Table A2.6: The session type information for session #3

This information shows that indeed only three unique OID prefixes have been found in this session. This information will in turn form a basis for the comparison of sessions in future algorithms. It should again be stressed that the port number information of the initiator (because of the operation type) may be disregarded in future comparisons between sessions, because the source port number changes per session for almost every manager implementation. This has been determined through manual observations made in various trace files from different locations.

When looking at the results of this algorithm for each of the handled sessions, it can be stated that none of these three sessions have the same session type, even if the port numbers are not considered. One of the reasons is that all resulting session types have different lists of found OID prefixes. This field, amongst others, must be equal, as a result of the complete definition of a session type.

A3 Example algorithm execution – session set detection

Although the description of the third algorithm and its steps has been explained alongside an example, that example was very abstract and straight forward. It is the purpose of this subsection to suggest and handle a more difficult input of sessions and their session types.

Since real traces do not give away the programmed behaviour details of the network elements involved in the recorded traffic in that trace, nothing will be given here in advance. The result should not be influenced by this.

The following limits will be stated in advance. Note that these values may be changed to the specifics of a trace or even per considered initiator, as has been stated in the algorithm description.

Limit	Value
Border session scan range	20 seconds
Timeout within session set	10 seconds
Min. session type occurrence	3
Max. session set length difference	50%

Table A3.1: Pre-defined limits

Then, consider the following table of session type occurrences that have been recorded in an artificial trace. This table will subsequently be given as input to the algorithm.

Start Time (sec.)	Session type	Start Time (sec.)	Session type
0,1	get-next-request-A[]-C[161]-[0]	704,5	get-request-B[]-C[161]-[0]
0,4	get-next-request-A[]-D[161]-[0]	751,9	trap-C[]-A[]-[0]
49,7	get-request-B[]-C[161]-[0]	889,2	get-next-request-A[]-C[161]-[1]
55,6	trap-C[]-A[]-[0]	894,8	get-next-request-A[]-D[161]-[0]
290,1	get-next-request-A[]-C[161]-[1]	1095,2	get-request-B[]-C[161]-[0]
294,4	get-next-request-A[]-D[161]-[0]	1290,6	get-next-request-A[]-C[161]-[1]
590,3	get-next-request-A[]-D[161]-[0]	1296,1	get-next-request-A[]-D[161]-[0]
595,0	get-next-request-A[]-C[161]-[1]	<< RECORDING ENDED AT 1400 SEC. >>	

Table A3.2: Possible input to this algorithm

Although this is a very short trace, it will highlight some of the problems that have been discussed in the previous subsections.

The first step, after this has been given as input to this algorithm, involves the removal of potential incomplete sessions at both borders, as a result of starting and stopping the recording of traffic. A limit for scanning for incomplete sessions is preset to 20 seconds. Scanning for sessions in this trace that occur within 20 seconds of a border results in only two sessions at the beginning of the trace that could possibly be incomplete. However, only the first of these two sessions is of a session type that is not seen anywhere else in the trace, therefore only the very first listed session shall be removed, before continuing with the next step.

The next step is the selection of one of the initiator IP addresses. Since there are three different initiator IP addresses (*A*, *B*, *C*), *A* shall be the first initiator IP address for consideration. This results in the following subset of sessions of the trace, which have been initiated by *A*:

Start Time (sec.)	Session type	Start Time (sec.)	Session type
0,1	<< NOT CONSIDERED >>	889,2	get-next-request-A[]-C[161]-[1]
0,4	get-next-request-A[]-D[161]-[0]	894,8	get-next-request-A[]-D[161]-[0]
290,1	get-next-request-A[]-C[161]-[1]	1290,6	get-next-request-A[]-C[161]-[1]
294,4	get-next-request-A[]-D[161]-[0]	1296,1	get-next-request-A[]-D[161]-[0]
590,3	get-next-request-A[]-D[161]-[0]	<< RECORDING ENDED AT 1400 SEC. >>	
595,0	get-next-request-A[]-C[161]-[1]		

Table A3.3: Subset of the given input

Step three involves the counting of the session types now listed. This results in an occurrence count of five for *get-next-request-A[]-D[161]-[0]* and a count of four of *get-next-request-A[]-C[161]-[1]*.

Then, the largest group of equally few occurrences should be selected out of these two session types. Clearly, this would only result in the session type *get-next-request-A[]-C[161]-[1]* to be selected, since this is the only one that is of the smallest number of occurrences.

In step 4 and 5, all possible potential session sets shall be discovered, which will form a basis for step 6. Following is the overview of the found potential session sets:

Start Time (sec.)	Session type
290,1	get-next-request-A[]-C[161]-[1]
294,4	get-next-request-A[]-D[161]-[0]

Table A3.4: Session set #1

Start Time (sec.)	Session type
590,3	get-next-request-A[]-D[161]-[0]
595,0	get-next-request-A[]-C[161]-[1]

Table A3.5: Session set #2

Start Time (sec.)	Session type
889,2	get-next-request-A[]-C[161]-[1]
894,8	get-next-request-A[]-D[161]-[0]

Table A3.6: Session set #3

Start Time (sec.)	Session type
1290,6	get-next-request-A[]-C[161]-[1]
1296,1	get-next-request-A[]-D[161]-[0]

Table A3.7: Session set #4

Session sets in tables A3.4 through A3.7 all contain one occurrence of the core session type each. In step 6, the best group of the involved session types shall be determined. In this case, the question reduces to whether only the core session type or also the other session type should be considered in the session sets that are going to make up a single session set type. In the mentioned four session sets, both session types occur equally often. As stated in the specification of this step in the algorithm, both session types shall be considered. This shall be taken to the next step: step 7.

Step 7 involves the finding of the session sets that are going to make up a session set type. This will result in this case in the listed four session sets in the tables A3.4 through A3.7. The algorithm returns to step 8,

because there is still one session that belongs to initiator *A* and because there will not be any more session sets that are undefined at this moment. This single session will simply result in the following floating session set, because it involves just one session set, which is lower than the stated limit.

Start Time (sec.)	**Session type**
0,4	get-next-request-A[]-D[161]-[0]

Table A3.8: Floating session set

There are no other sessions remaining which belong to initiator *A*, after this session set has been added to the list of floating session sets. This means that the next step for this algorithm is step 8, in which the just created floating session set is considered.

Step 8 defines the attempt to assign a floating session set to an existing session set type, or if that fails, a new session set type will be created. In this case, only one session set type exists for this initiator IP address, so only this single session set type will be considered. The pre-set limits suggest that the minimum percentage of comparable session types between a floating session set and a session set type must be 50%, or better. In this case, the floating session set contains just one session of a particular session type that can also be found in the session sets of the session set type. As a result, there is a 50% match. The next limit to be considered is regarding the length of the session set. This may only differ 50% of the session sets already listed of the session set type (i.e. a floating session set may be of length one, two or three in this case). Since also this prerequisite is met, this floating session set can be added to this session set type, which would yield, after considering this single initiator IP address, one session set type consisting of five session sets of which four are of length two and one of length one. No merging attempts shall take place in this case, because there is just one session set type.

Next is step 2 again, in which the next initiator IP address (*B*) is considered. Following is the overview of the sessions that are initiated by IP address B:

Start Time (sec.)	**Session type**
49,7	get-request-B[]-C[161]-[0]
704,5	get-request-B[]-C[161]-[0]
1095,2	get-request-B[]-C[161]-[0]

Table A3.9: Subset of the given input

There is just one session type involved here. Also, all sessions are more than the stated timeout value removed from each other, thus in this case, this would result in a single session set type consisting of three session sets which all contain one session type occurrence. Therefore, no floating session sets need to be considered. The next and final initiator IP address can now be considered.

The last initiator IP address that needs to be considered is initiator IP address *C*. After applying the filter for this initiator, this yields the remaining set of sessions:

Start Time (sec.)	**Session type**
55,6	trap-C[]-A[]-[0]
751,9	trap-C[]-A[]-[0]

Table A3.10: Subset of the given input

By the same reason used for initiator IP address *B*, it can now also be concluded in this case that there will be a single session set type, consisting of two session sets, because the two occurrences of the single session type are more than the preset timeout value removed from each other. Also, in this case no floating session sets shall be considered or processed.

After this, all sessions have been assigned to a single session set that belongs to a particular session set type. Then, the algorithm will go from point 8 to point 9, in which it will terminate and return the resulting three session set types that have been found.

This example has shown that both clearly periodic and aperiodic session set types can be detected. Also, it is shown that in the case of regular polling, as in the case of initiator IP address *A*, all polling instances match a session set that is part of a session set type. Subsequently, the respective session set type and its session sets cover the programmed polling settings of that initiator. In other words: all traffic related to a particular polling setting has been isolated into session sets that are of a single session set type. It is also shown that sessions which occur irregularly, as in the cases of the other two initiators, the algorithm is still able to identify the individual and independent session set types. This shows that this algorithm operates completely independently of any possible periodicity of session sets. The determination of intervals and whether or not the session sets of a session set type are periodic or aperiodic and if periodic, what the intervals are, are topics that will be addressed in the fourth algorithm description.

A4 Example algorithm execution – determination of periodicity

Although the algorithm description was accompanied by an example, that particular example was very straightforward and did not cover the more complicated issues that were discussed earlier in chapter 5.2. This subsection discusses a more complicated example of a session set type, consisting of both periodic and aperiodic traffic session sets.

Consider the following 19 session sets, which are all of the same session set type, given as input to this fourth algorithm.

Start Time (sec.)	**Last session start (sec.)**	**Last request (sec.)**	**Last response (sec.)**	**Number of sessions in session set**	**Session set**
10,58	12,27	12,78	12,88	4	Session set #1
25,62	31,59	32,30	32,35	10	Session set #2
124,84	129,99	130,52	130,62	10	Session set #3
225,89	231,68	232,56	232,66	10	Session set #4
310,48	315,86	316,73	316,83	10	Session set #5
325,32	330,76	332,22	332,32	10	Session set #6
380,97	386,12	387,53	387,63	10	Session set #7
424,13	429,73	431,04	431,14	10	Session set #8
525,27	530,66	531,65	531,75	10	Session set #9
609,97	615,53	616,26	616,36	10	Session set #10
625,18	630,92	631,58	631,63	10	Session set #11
725,03	730,53	731,60	731,70	10	Session set #12
825,38	830,92	832,27	832,37	10	Session set #13
910,14	915,22	916,23	916,33	10	Session set #14
1125,29	1130,33	1131,09	1131,14	10	Session set #15
1210,53	1216,19	1217,67	1217,77	10	Session set #16
1225,85	1231,40	1232,22	1232,32	10	Session set #17
1326,24	1331,29	1332,17	1332,22	10	Session set #18
1425,42	1431,97	1432,59	1432,68	10	Session set #19

Table A4.1: Set of session sets in a session set type given as input to this algorithm

The following list of values shall be used, besides this list of session sets given as input to the fourth algorithm. This list shall be followed by the description of the steps taken by this algorithm.

Limit	**Value**
Max. start session set length difference	25%
Max. interval deviation	10 seconds
Max. avg. interval deviation	3 seconds
Min. nr. of session sets in subset	4
Max. time between session sets	650 seconds

Table A4.2: Preset values used in this example

The very first thing that occurs is the removal of the incomplete first session set in the first step of the algorithm. Since the maximum start session set length difference is 25% and the average session set length of the session sets, minus the first and the last session sets is 10. This gives a minimum session set length of 7,5 sessions for the first session set. However, there are only 4 sessions assigned to the first session set, so the first session set shall not be considered further.

The next steps involve the detection of intervals and the sessions that contribute to these intervals respectively.

- First, an attempt is made to find an interval based on the combination of session sets 2 and 3.
    - After calculations, follows that the interval between these two is 99,22 seconds, based on the first timer reset point;
    - Using the formulas given in the description of step 2c and the given preset values, it follows that the next session set should occur within the range [218,06; 230,06];
    - In step 2d it is concluded that there is indeed a session set within the stated range. Therefore session set 4 is added to the subset of session sets that contribute to the same interval.
- The next step is to find the fourth member. First, the algorithm returns to step 2b.
    - Here the interval shall now be recalculated for the three members. Now, the average interval is 100,14 seconds, based on timer reset point 1, which yields the smallest total deviation of 1,83 seconds from its respective interval;
    - This gives the range for the possible fourth member session set: [318,86; 333,20];
    - In step 2d it is concluded that there is a session set within the stated range. Therefore session set 6 is added to the subset of session sets that contribute to the same interval.
- This process repeats for session sets 8, 9, 11, 12 and 13. The algorithm returns to step 2b, after finding session set 13 to be contributing to this single interval.
    - Here the interval shall now be recalculated for the currently nine members. Now, the average interval is 100,08 seconds, based on timer reset point 1, which yields the smallest cumulative deviation of 4,62 seconds from its respective interval;
    - This gives the range for the possible tenth member session set: [915,46; 925,46];
    - In step 2d it is concluded that there is no session set within the stated range. Because the now created set of session sets contributing to this single interval is larger than the stated limit of 4, this set of session sets shall be saved in step 3.
- Now, the algorithm starts again in step 2a, but now with 9 fewer session sets.
    - In step 2a the first two session sets shall be selected: session set numbers 5 and 7;
    - The interval shall now be recalculated for these two members, which is 70,49 seconds, based on timer reset point 1;
    - This gives the range for the possible third member session set: [445,46; 457,46];
    - In step 2d it is concluded that there is no session set within the stated range. As a result, because the now created set of session sets contributing to this single interval involves just 2 which is fewer than the stated limit of 4, this set of session sets shall not be considered further. Instead the algorithm will go back to step 2a, where it will attempt to select a different pair of initial session sets.
    - In step 2a session sets numbers 5 and 10 shall now be selected as initial session sets that may contribute to a particular interval. This is still allowed, because these two session sets are less than 650 seconds removed from each other.
- Now, the algorithm will retry to find a third session set, but now with a different pair of initial session sets.
    - After calculations follows that the interval between these two session sets is 299,49 seconds, based on the first timer reset point;
    - Using the formulas given in the description of step 2c and the given preset values, it follows that the next session set should occur within the range [903,46; 915,46];
    - In step 2d it is concluded that there is a session set within the stated range. Hence, session set number 14 is added to the subset of session sets that contribute to the same interval.
- The next step is to find the fourth member for this second interval. First, the algorithm returns to step 2b.

    - Here, the interval shall now be recalculated for the currently three session sets. The average interval is 299,83 seconds, based on timer reset point 1, which yields the smallest total deviation of 0,68 seconds from its respective interval;
    - This gives the following range for the possible fourth member session set: [1201,65; 1218,29];
    - In step 2d it is concluded that there is a session set within the stated range. Therefore, session set number 16 is added to the subset of session sets that contribute to the same interval.
    - Now, the process is repeated, but no new session sets shall be added to this set anymore, because no new session sets shall be found. As a result, this set of 4 session sets is sent on to step 3, because this set matches the limit of 4 session sets. So, now two sets of session sets have been found, each containing session sets that contribute to a different interval.
- The algorithm now goes again back to step 2a.
    - In step 2a the first two remaining session sets shall now be selected: session sets 7 and 15. But, because the time between these two session sets exceeds the limit of 650 seconds, session set 7 shall automatically not be considered further. This means that session set 7 will be marked as aperiodic. Now, the next first two session sets will be selected: session sets 15 and 17.
    - After calculations, it follows that the interval between these two is 100,56 seconds, based on the first timer reset point;
    - Using the formulas given in the description of step 2c and the given preset values, it follows that the next session set should occur within the range [1320,41; 1332,41];
    - In step 2d it is concluded that there is a session set within the stated range. Therefore, session set 18 is added to the subset of session sets that contribute to the same interval.
    - Subsequent execution of step 2b, 2c and 2d, shall result in the addition of session set 19 to this subset of session sets.

The algorithm has now allocated all given session sets to either a periodic set of session sets that contribute to a particular interval, or to the general set of session sets that are all marked aperiodic. In this case, two intervals have been found: 100 and 300 seconds. Also, one session set was marked aperiodic.

References

[1] Schönwälder, J. (2007). *Snmpdump*. https://trac.eecs.iu-bremen.de/projects/snmpdump (June 2007).

[2] Schönwälder, J. (2007). *SNMP Traffic Measurements*. Internet Draft. Bremen, Germany: Jacobs University.

[3] Ciocov, C. (2007). *Simple network management protocol trace analysis*. Bremen, Germany: Jacobs University.

[4] Harvan M. (2006). *SNMP Traffic Measurement and Analysis*. Bremen, Germany: Jacobs University.

[5] Grondman, I. (2006). *Identifying short-term periodicities in Internet traffic*. Enschede, The Netherlands: University of Twente.

[6] Stalling, W. (1998). *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Rev. 3. United States of America: Addison-Wesley.

[7] Oetiker, T. (2007). *MRTG*. http://oss.oetiker.ch/mrtg (July 2007).

[8] Broek, van den, J.G. (2007). *Bachelor of Science assignment support website*. http://wwwhome.cs.utwente.nl/~broekjg/bsc (July 2007).

[9] Schippers, J. (2007). *Analysis of SNMP usage in the real world*. Enschede, The Netherlands: University of Twente. p. 19

List of Figures

List of Tables