Efficient and scalable merging algorithms for wireless traces [*]

Bilel Ben Romdhanne Diego Dujovne
Thierry Turletti
INRIA, 2004 route des lucioles BP93, 06902 Sophia Antipolis, France

Abstract

Analysis of wireless experimentation results is a complex task to achieve. As multiple probes are needed in order to get a global view of a wireless experimentation, the resulting packet traces may be very large. In this paper we propose an algorithm that performs trace synchronization and merging in a scalable way. The algorithm output is stored in a configured MYSQL database allowing for smart packets trace storage. This solution reduces processing time by 400% and storage space by 200% with regard to raw trace files solutions.

1 Introduction

The Internet architecture and network protocols are evolving to accommodate for future user needs. Given the complexity and the size of current protocol stacks, any innovation or change must undergo several validations steps that may have different realism levels (analytical proofs, simulations and experiments). Even if modeling and simulation are important in the evaluation process, experimentation represents a major technique that should be undertaken before production solutions. However, experimentation is still complex and difficult to achieve.

The experimentation process must include several steps: scenario description, running, monitoring and capture, archiving and analysis [1]. From these steps one of the most important and complex is experiment monitoring and capture because it creates a high processing load and large data size. There are many proposals to get around these two problems. In wired networks, sampling allows reducing processing and memory resources, while in wireless networks, metrics computed on the fly simplify the monitoring and reduce memory requirements. Nevertheless, in both cases, understanding the experiment is limited and may include a ”margin of error” . To provide more complete experimentation understanding, monitoring should include full packet capture with the cost of an increased processing complexity [2]. Packet capture is typically done in several places and this increases the size of traces to be stored [3].

In fact, an important point that contributes to the difficulty of the experimental approach is the large amount of information generated through monitoring and the difficulty to merge this information in an easily exploitable engine. Accelerate data access becomes as important as capturing it, since the goal is to analyze and study the experiment results. Another important point is the impact of monitoring and capture during the experiment. Passive monitoring[1] requires several passive probes but traces generated with this method need to be synchronized before merging due to absence of a central synchronization point.

Merging traces represents a complex problem especially in wireless experimentations, due to packet redundancy in multiple probes. Merging traces solutions need to be efficient in order to process the large amount of generated traces. These solutions should provide an output data structure that allows easy and fast analysis and must be scalable in order to be used in large and various experimental settings. In addition, merging solutions need to include a specific collecting traces method in order to automate the whole process. We use the term ”pre-processing” throughout the paper to designate all the required opera-

[*] The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n 224263.

[1] Passive monitoring is a technique used to capture traffic from a network by generating a copy of that traffic

tions for ”merging” packets trace (i.e., synchronization, traces collection and generation of a unique trace).

In this paper, we propose a scalable and efficient pre-processing algorithm for wireless traces. Section 2 provides a survey of existing traces synchronization and merging efforts. Section 3 describes our pre-processing solution. Section 4 presents presents the performance results of our algorithm and finally section 5 concludes the paper.

2 Related work

Trace synchronization and merging was first investigated by Agrawala et al. in [3] where they show that several probes are required to obtain a global view of the experiment in wireless environment. To synchronize traces, they propose the use of beacon timestamps as a common reference and perform linear regression to fit all the traces. This method requires merging from all the traces simultaneously, thus generating a high pre-processing overhead.

Another solution that covers this gap is JIGSAW [4] which provides a single and unified view of all radio traffic on a 802.11 wireless network. JIGSAW is based on three axes: large-scale synchronization, frame unification and radio reconstruction. Its output is a data structure called Jframe, which includes source information of the packets. Although JIGSAW uses a database engine to synchronize and merge traces, its output format still keeps the content of the packets as a payload block and does not include multi-protocol features.

On the other side, current experimental platforms as PLANETLAB and ORBIT do not include trace merging as part of the experimental process. One of the most evolved wireless experimental platforms is ORBIT, a grid of wireless nodes with experimental control infrastructure. ORBIT provides real-time insertion of data into a database through the OML [2] framework. OML is a measurement data collection and organization framework, which enables the experimenter to define the measurement points and parameters, collect and pre-process measurements, and organize the collected data into a single database. It is a real-time framework where measurements are executed on experimental nodes during experimentation; In these conditions it is very difficult to include trace capturing as a feature without disrupting the experimentation.

Another network testbed, called Emulab [5], introduces an efficient approach to optimize the hardware use. It provides several virtualization levels in order to share resources between users. Moreover, it ensures the connectivity between virtual components and the real ones. Although such a hybrid configuration can result in a powerful tool, there are no means for trace merging or synchronization provided by the platform itself, and data must be processed with external tools.

In synthesis, capturing and merging traces allows to get a global understanding of network experimentation. Merging traces is done in physical layer of wireless networks but it is difficult in upper layers due to the high computing and memory resources needed to achieve this type of measurements. Captured traces generated from a single probe can represent a large amount of data, which increases the overall data size when we use several probes. In this case it is more efficient to store the traces within databases [6] (data files can be an archiving solution but does not allows efficient post-processing).

3 Efficient pre-processing data solution

In this section, we describe first the synchronization and merging algorithm, which reads the trace files, unifies the timestamps and then stores the packets into a database (section 3.1). Then, we propose an original approach to store and filter packet traces (subsection 3.2). After that we introduce the basic traces collection model(section 3.3), we discuss its weaknesses in order to introduce optimizations that we implemented (subsection 3.4), which further allows us to analyze the scalability of the system.

3.1 Synchronization and merging algorithm

The main difficulty in the merging process, is to identify each packet individually. This operation requires spatial and temporal characterization. Spatial information is provided by different localizations of the wireless probes, while temporal information is provided using timestamps generated by each probe. When probes are in monitor mode[2], each of the probes will use a different time reference to generate its trace (unlike the station mode where beacons generated by the AP can synchronize the clocks of all stations).

In order to synchronize the traces generated by the different probes, we propose to use the beacons generated by the AP as the time reference. We propose to use the beacons generated by the AP as the time reference. Beacon frames are transmitted periodically by the AP. The beacon frame carries the AP internal clock reference as part of the payload. When a probe captures a beacon frame, the wireless driver adds a PHY level header called *radiotap*, which includes the local time-of-arrival timestamps of the probe. The saved output is a packet with both the reference timestamps from the AP and the local timestamps, generated by the wireless card. We use this frame to determine the difference between the reference time and the local one. This difference is called the drift clock. The next step is to shift the local timestamps in the traces using the drift clock. The drift clock is not constant during the experiment, so we need to reevaluate it for each new beacon frame. We use the latest value each time we need to correct packet timestamps in order to synchronize the rest of the packets on the trace[3].

The goal of the synchronization algorithm is to prepare traces to merge. The merging process must identify each packet in each trace in a non ambiguous way. Due to the spatial distribution of probes, several packets may be received at the same time by different probes. We need to get around this ambiguity without overhead.

The identification algorithm is based on the following two-step criteria: First, we narrow the comparison to the nearby packets by defining a time window centered on the timestamp of the new packet[4]. Second, once we have obtained the list of pre-existing packets from the database within the time window, we use the hash of the payload of the new packet to check if it is already present in the database. There are two possible outputs of this process: If the new packet already exists in the database (i.e., it was inserted before from another trace) and in this case, we only keep the reference of the probe and the corresponding radiotap information. Else, the packet does not exist in the database and in this case, we proceed with the insertion of needed protocols information.

The whole process is executed as a continuous data flow algorithm. Raw traces are saved in very large files. As an example, a simple scenario including 6 probes for the monitoring of ping traffic between 5 stations for 300 seconds may generate 2.7GB raw data. Trace data is formated in XML (e.g. using tshark) in order to identify packet headers . Note that using XML simplifies the analysis of packets and allows processing a large number of protocols (about 750 protocols recognized by tshark). The merging software cannot load such a large file in memory to process it in batch mode. We have designed an algorithm called CrunchXML [7] that synchronizes and inserts packets into the database while reading data from the capture file. This algorithm reads the database structure and uses a state-machine in order to interpret the data from the capture file. With this algorithm, t only a small amount of temporary memory is required since only one packet is processed at a time.

This synchronization method is valid for any environment that uses the beacons synchronization solution as it is defined in the IEEE 802.11 specification (even for multiple AP testbeds). Another important advantage of this method is that user does not need specific equipments.

3.2 Smart packets traces storage

To insert packet traces into a database, we need an adequate data structure in order for the analysis phase to be easy and efficient. Note that relational database engines are based on tables, rows and columns. On the other hand, packets carry a number of protocol headers which is not known a priori. The simplest database schema is to associate one table to each protocol. Because the order of headers may change from one packet to the other, we stored each packet in a chained row structure.

The chained row structure stores a packet into the database by relating rows from tables (see Figure 1). Ev-

[2] In most of the cases, probes use the monitor mode, which allows packets to be captured without having to associate with an access point or ad-hoc network first.

[3] The maximum difference between two successive values thumbnail in experiments is included in the margin of error in the time windows (see section 3.3).

[4] The time window is bounded by $100\mu s$, which covers possible errors int the synchronization. The $100\mu s$ period is shorter than the length of the shortest packet on the 802.11 MAC protocol. This corresponds to a CTS or an ACK frame, with a 14-byte length, without including the PLCP nor the preamble. These packets are transmitted at 1Mbps rate, which results in $112\mu s$ of packet duration.

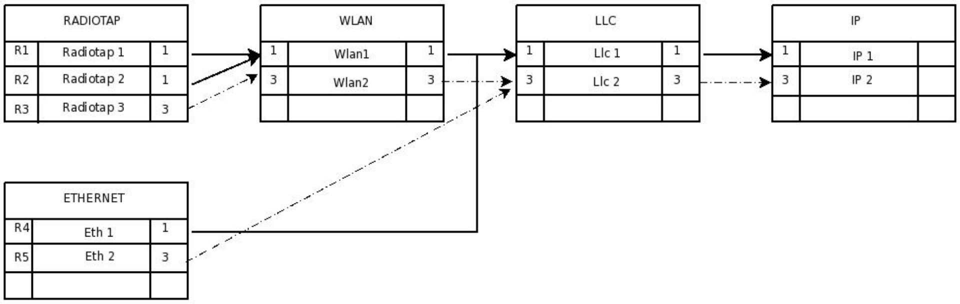

Figure 1: *Shortcut is used to access all the packet headers from the first one. This structure allows the user to access every information in two steps whatever the number of tables.*

ery packet is composed by several headers and each of them is related to a different protocol. In our database, we have created several tables, called protocol tables, which correspond to each of the protocols of interest for the experiment. Each of these protocol tables include the relevant protocol fields as columns and two special columns to recreate the header sequence from the packet. These two special columns point to the following protocol table and to the row where the next header of the packet is stored. Using this flexible structure, the database can hold any combination of protocol headers within a packet.

Furthermore, the storage space is reduced: First, common packets from the traces are stored only once, thus reducing considerably the used space. Second, only the protocols and fields under study are stored, so the payload and other protocols (unless strictly needed) are not part of the database. Indeed, when packets are separated in protocol tables, we can concentrate only on the protocol under study e.g. for counting, averaging or sampling field values. For example, if a general traffic analysis is desired, the throughput and airtime values can be calculated for the whole experiment using the radiotap table, which stores the timestamps, packet source and packet size.

Moreover, the chained row structure simplifies anonymization of traces. Each of the fields from the stored packets can be replaced with another data pattern through a database query, thus providing increased privacy for trace publication.

3.3 Traces collection

One of the key challenges in the design of an experimental large-scale testbed is how to collect experiment results from different probes in an efficient way. In fact, if we collect raw data in order to process them in the server, the network will be overloaded due to the large data size. We designed a distributed packet pre-processing model in which probes are used to insert packets into the database. Each of the probes acts as a client to the database server, which receives queries to search through the protocol tables and to insert rows according to the algorithm described in section 3.1.

Our insertion process is based on a data flow model which allows to process large trace files. The first prototype implements the routine described in Figure 2, for each packet individually. This basic model proves that the merging algorithms is a correct solution to provide a unique database trace but it did not provide satisfactory performance so we proceeded to design several optimizations that we describe hereafter.

We have implemented and tested this basic model and its performs correctly (no synchronization error). However, the scalability of the algorithm was poor in term of CPU requirements. So, we have proposed and implemented a set of optimizations presented in the following subsection.

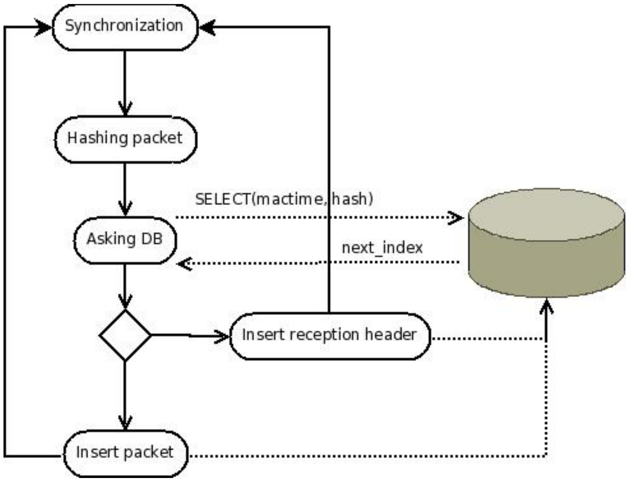

Figure 2: *Merging and synchronization algorithm*

3.4 Optimizations

The set of optimizations include five items: the Packet insertion transaction, Multiple Transactions Grouping, Concurrent Clients Access, Multiple Selection Grouping and an Adaptive Time Window.

Packet Insertion Transaction This optimization avoids sending multiple queries to the database in order to insert one packet and regroups them in one transaction. We minimize the global traffic and time needed to achieve packet insertion . To analyze the performance of this optimization, we use the following parameters: N the average number of affected protocols tables, S the average size of all fields for one header, T the average latency between the client and the server. With the basic data flow model described in 3.3, the time required to perform the operation is $T \times N$ and the traffic size is $(S + request\_overhead) \times N$. Using this optimization reduces the transaction time to T and the traffic size to $S \times N + request\_overhead$[5].

Multiple Transactions Grouping The new generation of relational databases offers the possibility to achieve multiple transactions in one request [6]. We use this feature to send more than one transaction per request. Let us define M, the number of transactions in one request. The traffic size will be $M \times N \times S + request\_overhead$ instead of $(S + request\_overhead) \times N \times M$ if the transactions were done without any grouping.

Concurrent Clients Access To take advantage of multiple transaction grouping, the packet insertion transactions have to be buffered and are delayed. Since multiple clients are accessing the database, concurrency problems may arise when more then one probe try to insert the same packet into the database. We have used the reference time (presented in section 3.1) to define an order between different packets and we take advantage from this order to provide a "locked by zone" feature. When a probe groups packets in order to insert them together, it must specify the minimum and the maximum timestamps in the grouped packets and must locked the access to the zone limited by these values in order to forbid packets duplication. This operation is named locked by time window. If another probe needs to use the locked zone, it must wait until the first probe concludes its packets insertion operation. This avoids inconsistencies due to packet diplucation in the database.

Multiple Select Requests As shown in Figure 2, the insertion decision is preceded by a specific request (called select) to the database to check if the actual packet exists. In the basic model, we must send individually this request for each packet because the global state of the database can change between two requests. But the previous optimizations introduce a new information: The state of all packets whose timestamps are include in the time windows can not change until the client releases the locked zone. In Figure 2, we can see that to make a decision, the client needs to send two arguments: the timestamps and the hash of the packet. If the packet exists, the database returns the next_index. Then, the client uses this index to chain the reception header to the packet in the database. This optimization consists on requesting three fields (timestamps, hash and next_index) from the database for all packets whose timestamps are in the time window. When the client processes captured packets,the insertion decision can be delayed until the packet timestamps is out of the time window.

Adaptive Time Window To increase the efficiency of the database insertion process, the main control variable is the time window interval. To adapt this value to the network and server conditions, we use the server response time T_server. The goal is to minimize the probability

[5] Note that transaction ensures that either the whole operation is performed or all the changes are reversed to the former state.

of a bottleneck during the insertion operation. We calculate a T_server threshold in order to get the best performance of the database, so if the T_server increases, time window is decreased in order to reduce to database load. Inversely, if the T_server decreases, the time window is increased. There are many algorithms that allows to implement this idea. we have implemented a simple one based on a smooth reevaluation of the T_server and the time window is updated using proportionately inverse ratio to the T_server variation.

4 Software performance

In the previous sections, we have presented proposals to perform pre-processing of wireless captured traces. We also presented optimizations of the basic model in order to increase scalability. In this section, we study the performance of the pre-processing mechanisms; we first start with an initial analysis of the impact of each optimization. Second, we study the impact of the number of simultaneous and concurrent active clients on the effectiveness of the system. Finally, to cover a wide range of experimental platform conditions, we study the performance of the algorithm under different CPU power capacities.

4.1 Impact of different optimizations

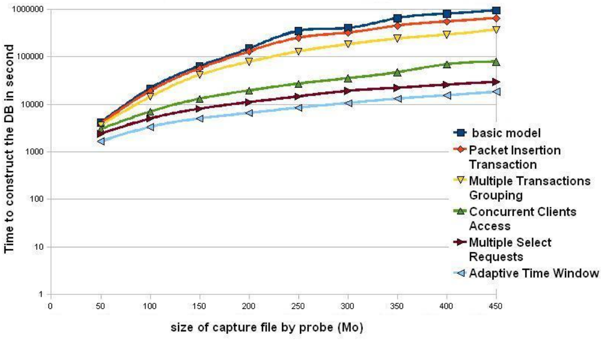

Figure 3: *Performance for each proposed optimization*

Our experimental scenario consists of 6 stations with generates packet trace sizes from 50 MB to 450 MB on each client. In the first approach, we distribute the insertion process from the probe stations towards a single database server. The stations are connected to the server through a 100Mbps switched Ethernet switch.

We can observe on Figure 3 that the non optimized solution, represented as the line with squares, has an non-linear performance and the whole process lasts more than 4 days to process 2.7 GB ($450MB \times 6$) of packet traces. When *transactional insertion* process is used, shown with the line with diamonds, the performance is improved by 50% but the non-linear behavior is still present. The other optimizations such as *multiple transaction grouping*, shown with downwards triangles, and *concurrent client access*, shown with upwards triangles, decrease the processing time, although it is still non-linear in both cases. The optimization that changes the insertion time from exponential to linear is the *multiple select requests*, shown with the righthand triangles. In fact, the SELECT operation is the most expensive in processing resources on the database server, and when the client increases the frequency of insert-selection operations, the database increases the time used to rebuild the table indexes. Nevertheless, when we use the *multiple select requests*, we minimize the CPU use at the server, which allows to execute indexing efficiently. Finally, the *adaptive time window* further improves this behavior by adapting the requests to the server capabilities.

4.2 Multiple Platform Scalability

To study the scalability of the optimized implementation in various platforms, we tuned the two following parameters: the *number of simultaneous clients* and the *available server_CPU power*. We change the number of the clients from 1 to 20 in order to cover a wide range of cases, while using a virtual machine as a server so that we can adjust its available processing power. The real machine is composed of two Xeon E5450 (Quad Cores) and 16 Gb of RAM. We increase the available processing power of the virtual machine as a percentage of the processing power of the server. For example, 25% means that the virtual server is equivalent to a dual-core processor with 4GB of RAM. Figure 4 presents the evolution of the average number of operations by second executed by the server during the insertion of 450 MB file. The higher the number of operations per time unit, the better.

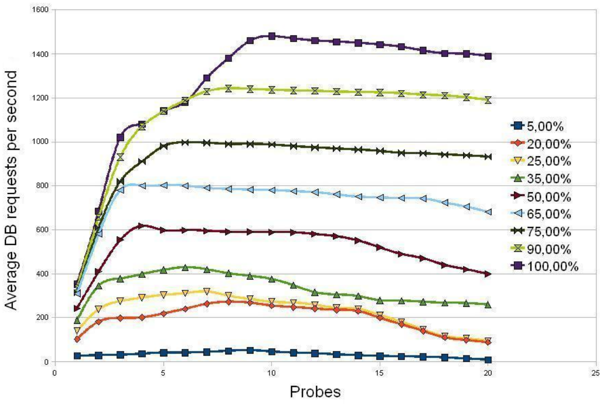

Figure 4: *Maximum number of request per unit time in function of the number of probes*

The first value for each curve represents the best value that can be reached by one client. In most of the cases, two stations perform as efficiently as half the processing time of a single station. For more than two stations, the performance depends on the available server processing power. For all cases we can distinguish two thresholds: Below the first threshold, the performance of the server increases proportionately to the number of stations. Between the two thresholds, the system is characterized by a stable performance, which is shared between different stations. From the second threshold, we observe that the performance decreases in function of the number of stations. We conclude that managing a high number of connections monopolizes much of the server CPU. These thresholds can be observed on each curve on Figure 4. For example, a quad-CPU (50%) provides the best performance until the limit of 4 simultaneuos stations, so the first threshold is 4 and share the max performance until 12 stations so the second threshold is 12.

5 Conclusion

In this paper, we considered the problem of merging captured traces and we have proposed an efficient solution for the merging and synchronization problem. Our solution called CrunchXML [7] is a distributed algorithm providing synchronization and merging of wireless traces and results insertion in a database. This solution aims to be scalable and efficient in order to be used in different experimental platforms. Performance of our solution is directly related to that of the database server. For this reason, we have analyzed the impact of server performance on the system stability. This analysis shows that our solution is adapted for most of the current infrastructure.

The efficiency of our solution allows us to study new domains such as packet tracing in multiple testbeds. The method we proposed is based on a database and this allows to identify packets individually in multiple traces. This approach can help researchers to study small flows in the Internet. Furthermore, our solution can be used to compare different validation approaches outputs (simulation, emulation and experimentation). Indeed, the unique hard constraint that each approach should respect is the use of real packet traces(e.g. NS-3 packets).

References

[1] D. Dujovne, T. Turletti, W. Dabbous. ”Experimental Methodology For Real Overlays”, ROADS’07, Warsaw,Poland, July 2007.

[2] Manpreet Singh, Maximilian Ott, Ivan Seskar and Pandurang Kamat. ”ORBIT Measurements Framework and Library (OML): Motivations, Design,Implementation, and Features”, Proceedings of IEEE Tridentcom, 2005.

[3] Jihwang Yeo, Moustafa Youssef, Ashok K. Agrawala, A framework for wireless LAN monitoring and its applications.

[4] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A. C. Snoeren, G. M. Voelker, and S. Savage. ”Automated Cross-Layer Diagnosis of Enterprise Wireless Networks” In SIGCOMM,2007

[5] M. Hibler R. Ricci L. Stoller J. Duerig ”Large-scale Virtualization in the Emulab Network Testbed ” USENIX Annual Technical Conference, Boston, MA, 2008

[6] M. Siekkinen, V. Goebel, and E. W. Biersack. ”Object-Relational DBMS for Packet-Level Traffic Analysis: Case Study on Performance Optimization. In Proceedings of IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services”, April 2006.

[7] http://planete.inria.fr/software/CrunchXML open Source under GPL2 license, 2009