Practical End-to-End Performance Testing Tool for High Speed 3G-Based Networks

Hiroyuki Shinbo, Atsushi Tagami, Shigehiro Ano, Toru Hasegawa, Kenji Suzuki

Practical End-to End Performance Testing Tool for High Speed 3G-based Networks

Hiroyuki Shinbo[1], Atsushi Tagami[1], Shigehiro Ano[1], Toru Hasegawa[1], Kenji Suzuki[2]

[1] KDDI R&D Laboratories Inc., 2-1-15 Ohara, Fujimino-shi, Saitama 356-8502, Japan
{shinbo, tagami, ano, hasegawa}@kddilabs.jp
[2] The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan
suzuki@cs.uec.ac.jp

Abstract. High speed IP communication is a killer application for 3rd generation (3G) mobile systems. Thus 3G network operators should perform extensive tests to check whether expected end-to-end performances are provided to customers under various environments. An important objective of such tests is to check whether network nodes fulfill requirements to durations of processing packets because a long duration of such processing causes performance degradation. This requires testers (persons who do tests) to precisely know how long a packet is hold by various network nodes. Without any tool's help, this task is time-consuming and error prone. Thus we propose a multi-point packet header analysis tool which extracts and records packet headers with synchronized timestamps at multiple observation points. Such recorded packet headers enable testers to calculate such holding durations. The notable feature of this tool is that it is implemented on off-the shelf hardware platforms, i.e., lap-top personal computers. The key challenges of the implementation are precise clock synchronization without any special hardware and a sophisticated header extraction algorithm without any drop.

Keywords: End-to-end Performance Tests, Clock Synchronization Protocol, Packet Header Analysis

1 Introduction

Providing high performances in the 3rd generation (3G) mobile systems which have begun being deployed is important for 3G mobile system operators. The 1xEV-DO (1x Evolution Data Only) system based on the specification of cdma2000 High Rate Packet Data [1] is an example of 3G mobile system and it uses a special data link protocol [1, 2]. The 3G mobile system operators (in the rest of paper, we call them just 3G operators) extensively perform interoperability tests [3], where end-to-end performances between a mobile terminal and a server are measured with various environments. In order to detect latent causes of performance degradations before a commercial service begins, testers

(persons who do tests) of the 3G operators usually do the two tasks. First, if performance degradation is observed at the tests, they identify which software at which network node is the cause of degradation. Second, testers check whether network nodes process a packet within the predefined duration or not. This testing is especially important for network nodes on the backbone because they should process it in a small duration, e.g., hundreds of micro-seconds, to provide high end-to-end performances.

Since network nodes' implementations are black-boxes to the 3G operators, the testers should capture packets at incoming/outgoing links of each network node with timestamps and then calculate how long each packet is hold at the network node by comparing the timestamps. However, this style of tests is very time-consuming because such calculations should be manually performed for all packets at all network nodes.

In order to automate such testers' tasks, we propose a multi-point packet header analysis tool which consists of IP packet header capture devices and the manager. An IP packet header capture device captures IP packets from a tapped link, extracts only IP packet headers and records them to its disk with timestamps. The manager does calculations which the testers manually do by controlling all the IP packet header capture devices.

The main goal is that it is implemented on an off-the-shelf hardware platform. Actually we use a lap-top PC (Personal Computer) with a crystal oscillator, network interface cards and a RAM (Random Access Memory) disk instead of using expensive commercial packet tester such as IXIA [4]. This style of implementation contributes not only to reducing costs of testing, but also to increasing chances of using such tools in various environments.

However, implementing it as user-space software on (lap-top) PCs is not trivial. There are two key challenges for the implementation. The first and the most difficult challenge is clock synchronization with scores of micro-second level precision. Clocks, of which values are used as timestamps, of all PCs should be synchronized to calculate how long an IP packet is hold by a network node. Since a maximum holding duration of some network node is less than several hundred milliseconds, i.e., about 400 micro-seconds, the maximum error among the clocks should be less than scores of micro-seconds. This precision is not easy to achieve without special hardware.

The second challenge is recording all IP packet headers to a RAM disk without any drop. Since an IP packet sent by either a server or a mobile terminal is encapsulated by PPP (Point-to-Point Protocol) and segmented by RLP (Radio Link Protocol) [1], the boundaries of captured packets do not always correspond to IP packet boundaries. Although the straight-forward way is to capture all packets in order to bridge the gap, it is difficult for PCs to record a number of 30 or 50 byte long packets segmented by RLP to its RAM disk without any drop.

In this paper, we have developed a multi-point packet header analysis tool which is useful for end-to-end performance tests over 3G mobile systems. The contributions of are twofold: a simple and precise clock synchronization protocol

and a real-time IP packet header extraction algorithm. First, taking advantage of the fact that clock synchronization need to be maintained only for a test duration which would be less than scores of minutes, we adopt a post-processing approach where timestamps are synchronized after all IP packet headers are records. This achieves simple, but precise clock synchronization with the maximum error of scores of micro-seconds. Second, we design a real-time IP header packet extraction algorithm specialized for 3G mobile systems by reading as small number of bytes from captured packets as possible. This algorithm enables to record all IP packet headers to a RAM disk without any drop. These sophisticated features make this tool so useful that it was used for commercial 3G packet services, such as 1xEV-DO and BCMCS (Broadcast-Multicast Service) system [5, 6]. In the tests with this tool, we detected more than 10 software bugs of network nodes, which were not detected by their vendors.

The rest of this paper is organized as follows: Section 2 describes the overview of 1xEV-DO system. Section 3 describes the overview of the multi-point packet header analysis tool. Section 4 and 5 describe the clock synchronization protocol and the IP packet header extraction algorithm, respectively. Section 6 describes how the tool is used at the actual tests. Section 7 describes related work.

2 Overview of 1xEV-DO System

Fig.1 shows network nodes and protocols of the 1xEV-DO system. The 1xEV-DO system provides a high speed IP communication between a mobile access terminal (called *AT* in Fig.1) and a server. We define that *Forward-link* is the direction from a server to an AT, and *Reverse-link* is the direction from an AT to a server. Each network node takes the following role:

- A server is located at the Internet.
- An HA (Home Agent) provides handovers between PDSNs according to the Mobile IP (MIP).
- A PDSN (Packet Data Serving Node) is used for authenticating/accounting ATs. It provides an endpoint of a PPP session between a PDSN and an AT.
- A PCF (Packet Control Function) / ANC (Access Network Controller) segments PPP frames to RLP (Radio Link Protocol) packets and reassembles RLP packets. RLP provides a reliable packet transfer using packet retransmissions.
- An ANTS (Access Network Transceiver System) transmits radio wave.
- An AT (Access Terminal) is a mobile access terminal.

In the 1xEV-DO system, PPP is used to encapsulate IP packets sent by either an AT or a server. We call these IP packets as end-to-end IP packets. PPP frames are segmented to 30 or 50 byte RLP packets between an AT and a PCF/ANC.

Fig.2 shows how end-to-end IP packets are segmented and reassembled by network nodes and how capsule-headers are used on the Reverse-link of the 1xEV-DO system. The packets are done so on the Forward-link. Encapsulation, segmentation and reassembly are performed as follows:

1. An application on an AT makes an end-to-end IP packet. The AT adds a PPP header to it and segments it to RLP packets.
2. The AT adds the RLP header to each segmented packet (S1 to S3), and transmits them to an ANTS at a radio link.
3. After each segmented packet is received, the ANTS adds a UDP/IP header and transmits it to a PCF/ANC.
4. The PCF/ANC removes the UDP/IP header and the RLP header. Then, the GRE/IP header is added to the segmented packets, and transmits the packets to the PDSN.
5. The segmented packets are reassembled at a PDSN. The PDSN also checks whether the IP packet is correctly reassembled or not by a CRC of the PPP header.
6. The PDSN adds a MIP header to the IP Packet and transmits it to a HA.
7. The HA removes the MIP Header and transmits the IP packet to a server.

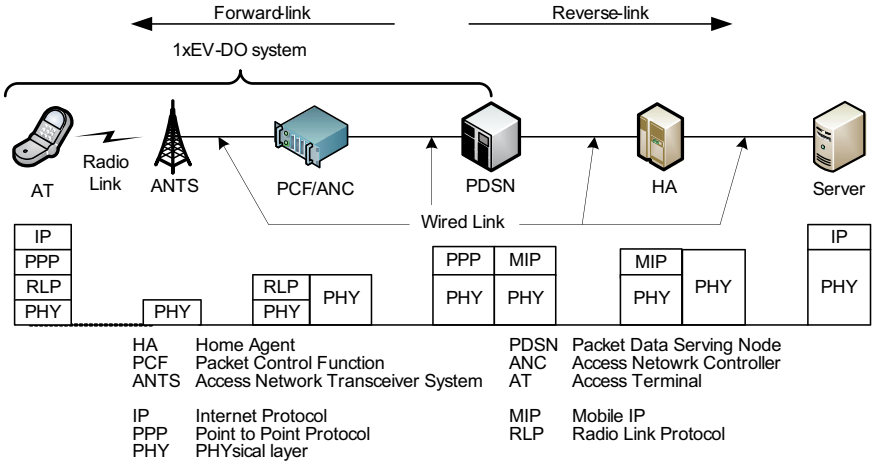

Fig. 1. Network Nodes and Protocols in 1xEV-DO system

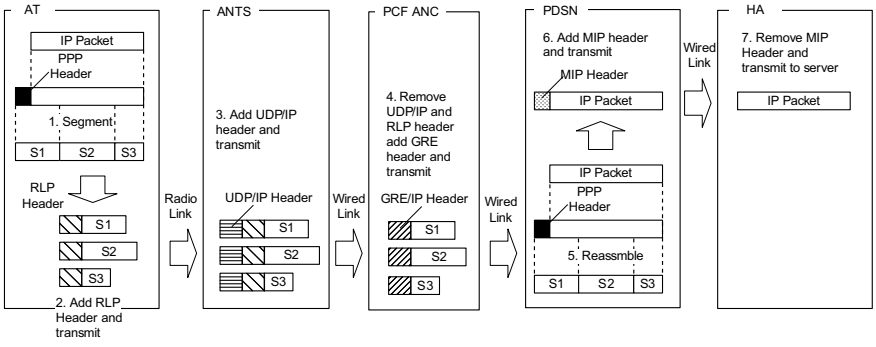

Fig. 2. Overview of segmenting and reassembling and used headers

3 Multi-point Packet Header Analysis Tool

3.1 Overview

The structure of multi-point packet header analysis tool is illustrated in Fig.3. It consists of *IP packet header capture devices* and the *manager*. In the rest of paper, we call the multi-point packet header analysis tool as the *analysis tool*, and an IP packet header capture device as a *capture device*. We implement them as the user-space software running a PC with a crystal oscillator, network interface cards and RAM disk.

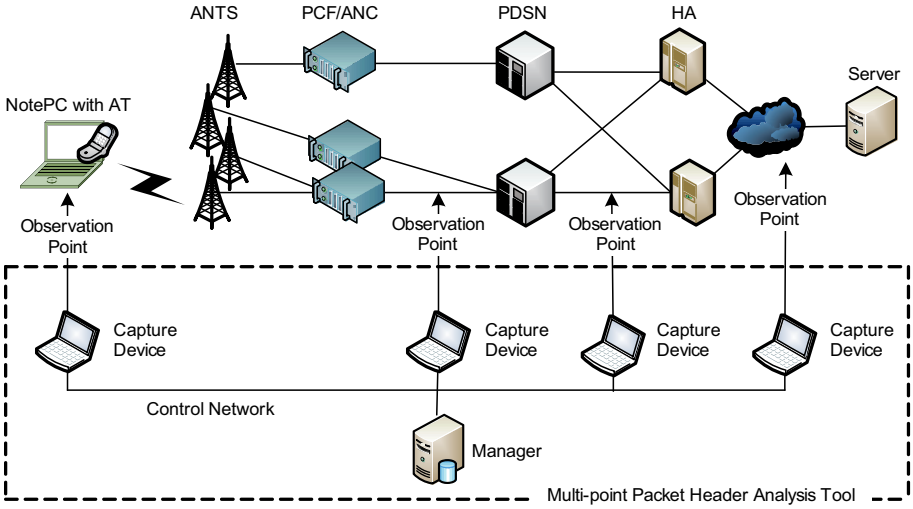

Fig. 3. Multi-point Packet Header Analysis Tool

Each capture device is a lap-top PC with two network interface cards. One network interface card is used to tap a link and the other is used to communicate with the manager. Capture devices are set to tap links between all pairs of network nodes including a server and an AT. We call tapped links as observation points. A capture device extracts an end-to-end IP packet header from a captured packet and records it into its RAM disk with a timestamp when the packet is captured. On the contrary, the manager controls capture devices and analyzes all captured IP packet headers collected from them.

3.2 How Analysis Tool is Used?

How this tool is used to analyze a communication flow between the AT and the server is as follows:

Preparation for Clock Synchronization
Before the AT and the server start communicating, the manager broadcasts a packet for synchronizing the capture devices' clocks to that of one the capture devices. All capture devices are connected via a broadcast medium such as Ethernet.

Packet Header Extraction
A tester makes all capture devices start capturing packets from tapped links. When a packet is captured, each capture device extracts an end-to-end IP packet header and then records it to its local RAM disk with its timestamp. The timestamp is read from the clock of PC which is a crystal oscillator. How the real-time IP packet header extraction algorithm handles protocol headers of captured packets will be described in Section 5.

Packet Holding Duration Analysis
After the test communication finishes, the manger collects all headers from all the capture devices. The timestamps set by individual capture devices are synchronized to the clock of one of the capture device. In other words, all timestamps are re-written so as to be synchronized to such device's clock. How clocks are synchronized will be described in Section 4. Then by analyzing timestamps for the same end-to-end IP packet at difference observation points, a tester calculates how long each end-to-end packet is hold by each network node.

4 Clock Synchronization Protocol

4.1 Problem Statement

The required precision should be around scores of micro-seconds. This is because the PDSN and the HA, i.e., backbone network nodes, handle a number of communication flows between ATs and servers. Usually, these network nodes should process each end-to-end IP packet within less than hundreds of micro-seconds. In some system, the average duration of processing a packet is about 400 micro-seconds as described in Section 6. Thus the goal of the maximum error is within scores of micro-seconds.

In order to precisely explain our clock synchronization protocol, we define several terms. A clock is a hardware register of a PC and its value is created from its crystal oscillator. A clock value is how many the crystal oscillator ticks and corresponds to the elapsed time. We use a time value and a timestamp interchangeably as a clock value. When a clock value is set to a captured packet, we call it is a timestamp. A clock frequency is how many it ticks in one second. Clock synchronization or synchronizing clocks means that difference between clock values of different capture devices are less than some threshold. For example, if the difference is always less than 100 micro-seconds, clocks are said to be 100 micro-seconds precise or the maximum error is within 100 micro-seconds.

4.2 Hurdles to Prevent Clock Synchronization

The goal of clock synchronization protocol is to synchronize all clocks of capture devices to a selected capture device, which we call it the *master capture device*, within scores of micro-second error. The protocol consists of the two procedures. First, the manager broadcast a packet informing all capture devices of clock values being set to 0 (initialization). During the test, each capture device reads its clock value and sets it as a timestamp to a captured IP packet header. Second, after the test, timestamps at different capture devices are compensated so that the clocks are synchronized.

There are two factors which prevent the clocks from being synchronized.

Long-term and Short-term Errors of Crystal Oscillators
A crystal oscillator is not either accurate or stable. Each oscillator's frequency is different from the ideal frequency. The difference is called the *frequency error* and that in most PCs is accurate to one part in 10^4 to 10^6. Besides, its frequency changes due to environmental factors such as variations in temperature and supply voltage. Due to frequency errors between two devices, the clock values are gradually drifting as shown in Fig.4. The x-axis and y-axis show the ideal time and the time observed at the ideal clock or the actual clock. The ideal clock means a clock without any frequency error. The dotted line shows how the ideal clock advances, and the angle is 45 degree. The observed time values of the actual clock are plotted and they are interpolated to the line using the least squares method. The angle difference between the two lines corresponds to the average frequency error. Since this angle or the difference is stable for a long duration, we call it a long-term error. On the contrary, the plots are not always on the interpolated line. A short term error is a difference between the observed time value of the actual clock and the dotted line.

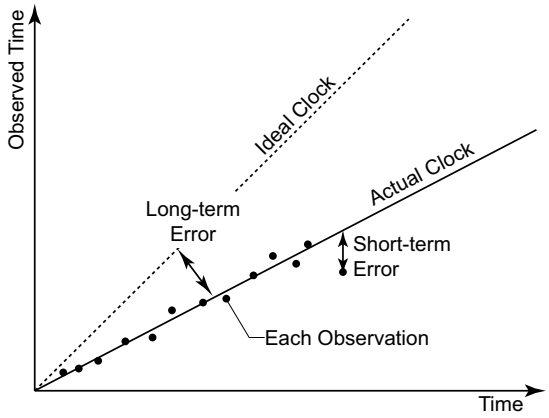

Fig. 4. How Ideal and Actual Clocks Advance

Non-deterministic Delay of Initialization Packet Transfer
At the initialization, the manger broadcast a packet for setting all clock values to 0. However, the packet does not always reach capture devices at precisely the same time. In other words, delay of sending the packet from the manager to each capture is not deterministic due to several non-deterministic delays: the time spent by the manager, the delay incurred waiting for access to a physical interface card, the time needed for a packet from the manager to a capture device, and the time required for the capture device's network interface card to receive and notify the user space software of its arrival. Due to such non-deterministic delays, the ideal time of capture device is different from that of the master capture device.

4.3 Design Principles

We set out the two design principles to compensate for the two factors.

Compensating for Clock's Long-term Error
We assume that the duration of each test is less than 20 minutes. We choose 20 minutes because it is recommended that TCP performance tests should continue more than 15 minutes. Due to such a short duration, we can assume that a hardware clock is stable for the duration and that we can ignore the short term error. By assuming that all clocks are stable, we can easily synchronize the clocks by compensating for the angle difference from the capture device to the master one.

Compensating for Non-deterministic Delay with Broadcast Communication
We compensate for non-deterministic delays using the broadcast media. Capture devices, which are usually located at an in-house test-bed, are directly connected each other via a broadcast medium so that a packet sent for synchronizing clocks are received by all the capture devices at almost the same time.

4.4 Preliminary Measurements of Long-term and Short-term Errors

How the design principles work well depends on how long-term and short-term errors of actual crystal oscillators are, how stable they are and how the packet delays of the actual broadcast medium are. Thus we have measured them in the following experimental conditions:

- A packet tester, e.g., *IXIA 400* [4], is used to broadcast a 40 byte long test packet every second for 20 minutes. The clock's precision of the packet tester is 1 PPM (Part Par Million).
- A shared Ethernet hub is used as a broadcast medium.
- Two PCs, which run Linux (kernel 2.4.18), are set at the broadcast medium and capture the test packets using *tcpdump* software [7]. Tcpdump runs in the user space and sets a timestamp to every captured packet.

– This 20 minutes experiment was performed 10 times and thus 20 measurement results for a clock were totally obtained.

Clock's Long-term and Short-term Errors
Two typical measurement results are shown in Fig.5. The x-axis is the clock value (the time value) at the packet tester and the y-axis is the error from the clock value at the PC (*PC1* or *PC2*) from that of the packet tester. We interpolate all the errors from the clock values to the lines with the least squares method. Among the 20 measurement results, 19 measurements show that the clock was stable for the 20 minutes. This result is shown by the line with caption *PC1* in Fig.5. All the errors are almost on the interpolated line. On the contrary, at one measurement, the clock was not stable, i.e., the clock frequency changes during the 20 minutes. This result is shown by the line with caption *PC2* in Fig.5. The angle of interpolated line of *PC2* changes when 445 seconds elapses at the packet tester. Since the error of *PC2* increases to about 300 micro-seconds after 20 minutes elapse, it is difficult to make the error at any time less than 100 micro-seconds only by compensating for the angles of the interpolated lines.

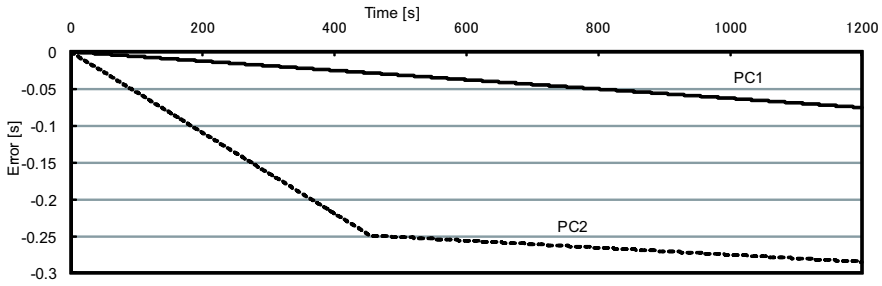

Fig. 5. Timestamps at the Two PCs

Due to this, we adopt an approach that results of a test during when such clock unstableness is detected are thrown away and the test is retried.

Besides, we calculate all the short-time errors, i.e., the distances from all the measured clock values to the interpolated lines. The maximum short-time error is about 20 micro-seconds. Since the long-duration error would become almost 0 by compensating for the angle differences, this 20 micro-seconds level error is not a problem to synchronize clocks within the maximum error of scores of micro-seconds.

Non-deterministic Delay
We cannot correctly measure how long the difference between the delays from the packet tester to each PC is. However, the above maximum 20 micro-seconds error includes both the short-term clock error and the above difference. Thus

it is estimated that the maximum difference is less than 20 micro-seconds. It means that the differences between start times at different PCs are less than 20 micro-seconds. As the result, the error of clocks would be less than twice as 20 micro-seconds, i.e., 40 micro-seconds during 20 minutes.

4.5 Protocol Details and Performance

This subsection describes the details of the clock synchronization protocol. It is assumed that there are n capture devices numbered from 0 to $n-1$ and the master capture device is 0.

1. The manager broadcasts m broadcast packets $P_s = \{P_{s1}, P_{s2}, \ldots, P_{sm}\}$ to all capture devices at equal intervals via the broadcast medium. Each capture device k records the clock values when it received P_s as $t_s(k) = \{t_{s1}(k), t_{s2}(k), \ldots, t_{sm}(k)\}$.
2. When the test starts, the capture device k reads its clock value and records it as a timestamp of captured IP packet header. The timestamp of the i th IP packet header is denoted as $t_k(i)$.
3. After the test finished, the manager broadcasts m broadcast packet $P_e = \{P_{e1}, P_{e2}, \ldots, P_{em}\}$ to all capture devices at equal intervals via the broadcast medium. Each capture device k records the clock values when it received P_e as $t_e(k) = \{t_{e1}(k), t_{e2}(k), \ldots, t_{em}(k)\}$.
4. The manager calculates angles of the clock values before and after the test, i.e., $t_s(k)$ and $t_e(k)$, using the least squares method. If the difference between the calculated angles is larger than the predefined threshold th such that $Angle(t_s(k)) - Angle(t_e(k)) \geq th$, the frequency error of capture device i is larger than the predefined threshold. The clock of capture device k cannot be synchronized to the master capture device 0. The clock synchronization protocol stops. (As the result, the test result would be thrown away and this test would be re-tried again.)
5. The manager rewrites timestamp $t_k(i)$ to $\overline{t_k}(i)$ according to Equation(1), and $t_{s1}(0)$, $t_{em}(0)$ are the clock values of the first and the last packets received by the master capture device. (All timestamps of capture device k are synchronized to those of the master capture device 0.) This rewrite is performed for all the capture devices except for the master capture device 0.

$$\overline{t_k}(i) = a_k \cdot t_k(i) + b_k \tag{1}$$

where

$$a_k = \frac{t_{s1}(0) - t_{en}(0)}{t_{s1}(k) - t_{en}(k)}, b_k = \frac{t_{s1}(k) \cdot (t_{en}(0) - t_{s1}(0))}{t_{s1}(k) - t_{en}(k)}$$

We applied the clock synchronization protocol to the 10 experiments described in Section 4.4. *PC1* and *PC2* are regarded as a capture device and the master capture device, respectively. For the 10 experiments, the maximum error after clock synchronization is about 16.8 micro-seconds for 9 experiments.

At one experiment, the clock synchronization protocol stops. (This corresponds to the case that the clock frequency changes after 445 seconds elapse in Fig.5.) This result shows that our clock synchronization protocol can synchronize clocks within an error of scores of micro-seconds. This definitely fulfills the tool's requirement, i.e., the maximum effort within scores of micro-seconds.

5 End-to-end IP Packet Header Extracting Algorithm

5.1 Design Principles

It is not trivial to capture and record all packets in the 1xEV-DO system without any drop. We should be careful to handle packets at a link between a PCF/ANC and a PDSN. The PDSN receives packets with GRE/IP headers (we call these packets as GRE/IP packets) to which an end-to-end IP packet encapsulated according to PPP by an AT is segmented. Each size of segmented packet is either 30 or 50 bytes long as shown in Fig.2 of Section 2. It is difficult for an off-the-shelf PC with tcpdump software to record a number of small GRE/IP packets into its RAM disk at the link rate of 100Mbps without any drop. Since many flows between ATs and servers are aggregated at this link, the total traffic is about 100Mbps. Thus we decide to record only end-to-end IP packet headers instead of all packets at the link between a PCF/ANC and a PDSN. Please note that a timestamp at the link between a PCF/ANC and a PDSN is the time when the last GRE/IP packet is captured.

5.2 End-to-end IP Packet Header Extraction Algorithm

In order to record only end-to-end IP packet headers, we design a real-time header extraction algorithm. The algorithm is not trivial because boundaries between PPP frames which encapsulate end-to-end IP packets do not always correspond to the beginning/end of GRE/IP packets. Besides a PPP header does not have a PPP payload length and a size of a PPP frame is not the same as that of an encapsulated end-to-end IP packet due to PPP escape sequences.

The algorithm finds boundaries of PPP frames as small number of accesses to captured GRE/IP packets as possible. It makes use of the fact that an end-to-end IP packet header follows just after a PPP header and that the PPP payload is longer than the end-to-end IP packet by the size of the PPP escaped sequences. For example, PPP escapes 0x7E to two bytes 0x7D-5E because 0x7E is used as the Flag Sequence. Thus the algorithm skips reading bytes from the captured GRE/IP packets while the sum of skipped GRE/IP packet sizes is less than the end-to-end IP packet size, and then reads bytes from the next GRE/IP packet on byte-to-byte basis to find a flag sequence of the next PPP frame.

The details of the algorithm are as follows, as illustrated in Fig.6. The algorithm uses variable *REMAINING_BYTES* to skip reading the bytes described above.

1. The algorithm captures a GRE/IP packet.
2. Bytes of the captured GRE/IP packet are read on byte-to-byte basis until finding Flag Sequence (0x7E) of a PPP frame. After finding Flag Sequence, it goes to 3. Otherwise, it goes to 1.
3. "Protocol" field in the PPP header is checked. If "Protocol" field is IP packet, it goes to 4. Otherwise, since this PPP frame is not IP packet, it goes to 2 for finding the next PPP frame.
4. It obtains an end-to-end IP packet length by reading an IP packet length field of an end-to-end IP packet header just after the PPP header. If the current GRE/IP packet does not include the IP packet length field, the next GRE/IP packet is read on byte-to-byte basis until finding the IP packet length field. Then, the end-to-end IP packet length is set to variable *REMAINING_BYTES*.
5. When the next GRE/IP packet is captured, it checks the payload size of the GRE/IP packet. If the payload size is less than *REMAINING_BYTES*, the end of current PPP frame does not exist in this GRE/IP packet. In this case, *REMAINING_BYTES* is decreased by the payload size of this GRE/IP packet, and it repeats this procedure. Otherwise, since there is the end of the current PPP frame in this GRE/IP packet, it goes to 6.
6. Bytes of the captured GRE/IP packet are read on byte-to-byte basis until finding Flag Sequence as the end of the current PPP frame. Then, it puts the end-to-end IP packet header information with this GRE/IP packet timestamp to a RAM disk, and it goes to 2.

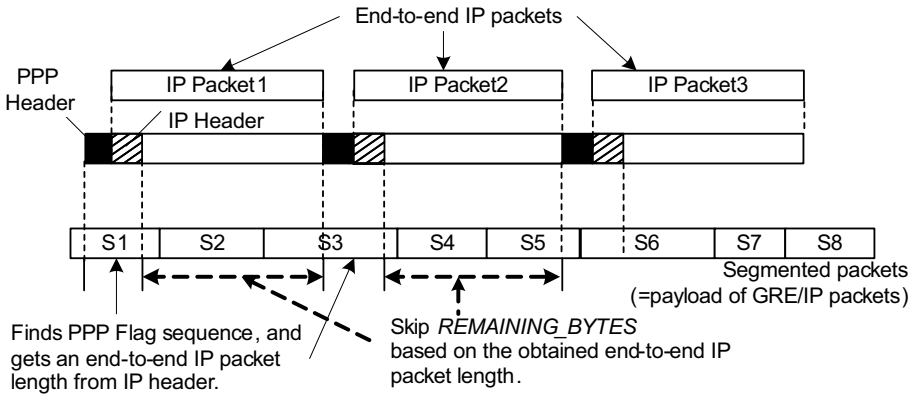

Fig. 6. End-to-end IP Packet Header Extraction Algorithm

We have measured performances of our algorithm recording only IP packet headers and the tcpdump software recording packets. The traffic is collected from the real 1xEV-DO system. *Tcpreplay* [8] is used to replay the collected traffic with various rates. The performance tests are performed using PCs with GbE (Giga

bit Ethernet) network interface cards. The number of transmitted packets is 7,119,000 (1,228Mbytes), which includes 1,547,000 end-to-end IP packets. Table1 shows the numbers of end-to-end IP packets correctly recorded by tcpdump, and the numbers of end-to-end IP packet headers recorded by our algorithm. The performance of our algorithm is better than that of tcpdump. Our algorithm did not drop any end-to-end IP header even for 120 Mbps traffic. On the contrary, the tcpdump software dropped some end-to-end IP packets.

Table 1. How many Packets/Headers are recorded?

Traffic Packet per sec	80Mbps 63kpps	100Mbps 79kpps	120Mbps 96kpps
tcpdump	1546963 (99.9%)	1539242 (99.50%)	1541213 (99.62%)
Our algorithm	1547000 (100%)	1547000 (100%)	1547000 (100%)

Note: The percentage means the ratio of how many packets (tcpdump) or headers (our algorithm) are recorded.

6 Actual Tests with the Developed Tool

We applied the analysis tool to tests of the BCMCS system which provides multicast packet delivery to ATs using forward-links of the 1xEV-DO system. The network nodes and protocol stacks are those of Fig. 1 except for that MIP and HAs are not used. Important requirements to the BCMCS system are that the number of lost packets should be as small as possible and that every packet should be processed by the PDSN within hundreds of micro-seconds. The main objections of these tests are to check whether the PCF/ANC and the PDSN satisfies the above requirements under the condition that radio-link's quality is good and that no congestion occurs at the backbone network.

We conducted about 50 test scenarios in this environment. At each scenario, a server sends multicast packets to ATs at various sending rates. In order to check how long each packet is process at network nodes, we set capture devices at the link between the server and the PDSN (*observation point 1*), the link between the PDSN and the PCF/ANC (*observation point 2*) and the AT (*observation point 3*).

During the tests, we found more than 10 software bugs of the PDSN and the PCF/ANC. The bugs are classified into two bugs. Fig.7 (a) and (b) show typical examples of these two. Please note that several packets are received by the AT at the same time in Fig.7 (a) and (b) because such packets are encoded using

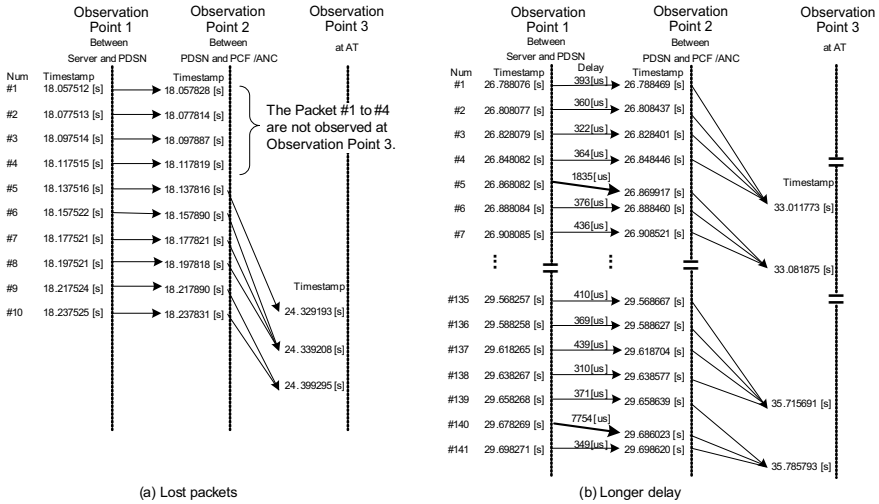

Fig. 7. Packet sequences with typical bug cases

some FEC (Forward Error Collection) method and these packets are decoded at the same time by the AT.

Lost packets at the PCF/ANC

At some scenario, we observed packet losses at the AT even if there is no congestion in the network. Fig.7 (a) shows the outputs of the analysis tool, i.e., the timestamps of the multicast packets (these are end-to-end IP multicast packets) at three observation points. We easily knew that the multicast packets were lost between the observation points 2 and 3. It means that they were lost either at the PCF/ANC or the ANTS. Then we identified that they were lost at the PCF/ANC by analyzing the communication logs of the PCF/ANC. Finally, after talking with its vendor, we found that the PCF/ANC had a software bug such that multicast packets are lost with a specific condition.

Longer Duration of Processing Packets at PDSN

At another scenario, the test seemed to be successful because all the multicast packets were correctly received by the AT. However, we observed two multicast packets (Packet #5 and #140 in Fig.7 (b)) went through longer delay than one millisecond between the observation points 1 and 2 although most packets did about 400 micro-second delays. We identified that the longer delay occurred at the PDSN and asked its vendor to identify the reason and fix the problem. Finally, we knew that this was caused by the software bug of how the PDSN handles packets at its memory.

Although the vendors intensively tested the software for the PCF/ANC and the PDSN, they missed these bugs. It means that the analysis tool was useful to

detect such bugs. Especially, in the case of Fig.7 (b), we would miss the software bug unless we used the analysis tool.

7 Related Work

As far as we know, there is no testing tool which captures packets or packet headers with synchronized timestamps at multiple observation points. Besides, there would be no study for extracting IP headers from packets which are transferred on 3G mobile systems.

On the contrary, clock synchronization is studied for distributed systems. The most straight-forward way is to use the Global Positioning System [9]. Many products are commercialized using the pps (plus per second) signal of GPS and synchronize their clocks with an absolute precision of better than 10us to the absolute time, i.e. UTC (Universal Time, Coordinated). However, the GPS requires a clear sky view, which is usually unavailable in our test-bed environments. There is another product, which synchronizes clocks with a precision of better than 1 nano-second using common timing signal [10] among nodes only which are located within a few meters. This requirement is not fulfilled in our test-bed environment and this product requires a special and expensive hardware.

Over the years, many clock synchronization protocols, which would be implemented based on the software, have been designed for distributed system [11–16]. NTP [11] and SNTP [12] are most prominent clock synchronization protocols used in Internet; however, their several millisecond level precision is not enough for our testing tool. IEEE1588 [13] is a clock synchronization protocol over local area networks. Although this precision is one micro-second level on some types of local area networks, it is vulnerable to random delay on local area network switches which would be used as the broadcast medium of our testing tool. Some clock synchronization protocol implementations [14, 15] achieve several micro-second precision. However, these optimize firmware of MAC (Media Access Control) protocol and require some hardware support.

Our clock synchronization scheme is similar to the clock synchronization protocol of [16] in that the both use the broadcast medium for compensating non-deterministic packet delays. However, taking advantage of the fact that clock synchronization needs to be maintained only for a testing duration, which would be just less than scores of minutes, we adopt a post-processing approach where the timestamps are synchronized after all IP packet headers are captured with timestamps. This makes our scheme far simple and efficient from the clock synchronization protocol of [16].

8 Conclusion

This paper has proposed a multi-point packet header analysis tool for end-to-end performance tests for 3rd generation (3G) mobile systems. This tool extracts end-to-end IP packet headers with synchronized timestamps at multiple observation points. The synchronized timestamps at multiple observation points enable

testers to identify a reason of performance degradation and to check whether each network node processes a packet within a predefined threshold. We implemented this tool on off-the-shelf hardware platforms, i.e., lap-top personal computers, which enable this tool to be used widely for various purposes. The notable features of this tool are scores of micro-second precision in clock synchronization without using any special hardware and a sophisticated end-to-end IP packet header extraction algorithm without any drop. Due to these features, this tool is so practical and useful that it was used for testing a few commercial 3G mobile systems. Especially, scores of micro-second precision in synchronized clocks was useful to detect more than 10 bugs of some network nodes which were not detected by their vendors. This was helpful to launch the commercial 3G services on time. Besides, although this tool is developed for testing 3G mobile systems, the main functions of multi-point packet capture and scores of micro-second level clock synchronization are so general that this tool is used for testing various IP-based communication systems.

References

1. 3GPP2: cdma2000 High Rate Packet Data Air Interface Specification. 3GPP2 Spec. of C.S.0024-0 v4.0 (2002)
2. 3GPP2: Wireless IP Network Standard. 3GPP2 Spec. of X.S0011-001-C v1.0 (2003)
3. ETS 300 406: Methods for Testing and Specification (MTS). Protocol and profile conformance testing specifications Standardization methodology. (1995)
4. IXIA, http://www.ixiacom.com/
5. 3GPP2: cdma2000 High Rate Broadcast-Multicast Packet Data Air Interface Specification. 3GPP2 Spec. of C.S.0053-0 v2.0 (2005)
6. Kyungtae K., Jinsung C. and Heonshik S.: Dynamic packet scheduling for cdma2000 1xEV-DO broadcast and multicast services. Proc. of Wireless Communications and Networking Conference 2005 (WCNC2005), Vol4, pp 2393–2399. (2005)
7. tcpdump, http://www.tcpdump.org/
8. tcpreplay, http://tcpreplay.synfin.net/
9. Elliott D. Kaplan, editor: Understanding GPS: Principles and Applications. Artech House. (1996)
10. National Instruments Corporation PXI System, http://www.ni.com/pxi/.
11. Mills D. L.: Internet Time Synchronization: Network Time Protocol. IEEE Trans. Communications, 39(10), pp 1482-1493. (1991)
12. Mills D. L.: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. RFC 2030, IETF. (1996)
13. IEEE1588: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. IEEE Standard 1588, 2002.
14. Liao C., Martonosi M. and Clark W.: Experience with an adaptive globally-synchronizing clock algorithm. Proc. of Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '99), pp.106–114. (1999)
15. Maroti. M., Kusy B., Simon G. and Ledeczi A.: The Flooding Time Synchronization Protocol. Proc. of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys '04), pp.39–49. (2004)
16. Elson J., Girod L. and Estrin D.: Fine-Grained Network Time Synchronization using Reference Broadcasts. Proc.of OSDI (Operating systems design and implementation) '02, pp.147–163. (2002)