BCMP Performance Test with TTCN-3 Mobile Node Emulator

Sarolta Dibuz, Tibor Szabó and Zsolt Torpis

Conformance Laboratory, Ericsson Hungary Ltd.
Laborc 1, HU-1037 Budapest, Hungary
{Sarolta.Dibuz, Tibor.Szabo, Zsolt.Torpis}@eth.ericsson.se

Abstract. In this paper we show guidelines for performance testing through *BRAIN Candidate Mobility Management Protocol (BCMP)*. At first, we investigate the main issues of our tests, then we describe briefly the applied TTCN-3 based distributed parallel test environment. We present the structure of the network to be tested and the Mobile Node Emulator, the main functional element of the performance test. We describe the actual test and the results. Finally, we summarize our experiences and proposals.

1 Introduction

Conformance testing of telecommunication protocols, services and applications has become an unavoidable process within a lifecycle of a product. To achieve high quality one has to ensure that a protocol implementation meets the requirements of the corresponding specification. Objectives, methods and tools of conformance testing are standardized, deeply studied and deployed. The most widely used conformance testing notation is TTCN version 3 [1]. There are some tools available to compile and execute TTCN-3 test suites against protocol implementations.

However, there are non-functional requirements such as performance, quality of service (QoS) and scalability characteristics of an implementation that are out of scope of conformance and functional testing. These requirements are as important as functional requirements. It is often needed to verify that a protocol, an equipment or a network meets non-functional requirements. The goal of performance testing is to ensure that an implementation can operate at a certain performance level. On the other hand, the purpose of scalability testing is to determine whether an application scales for the workload growth.

One of the main differences between conformance and performance testing is the different interpretation of time. Timers in a conformance test ensure that an event occured too early, too late or not at all. In performance testing, one has to establish streams of data, not separated protocol events. Correct timing of the generated load becomes very important.

For performance and scalability testing the conformance of the implementation is assumed. However, since high load may affect correct functionality of the

implementation, it is necessary to track functional correctness during a performance test [2].

2 Related Work

There are numerous contributions to the field of performance testing with TTCN. PerfTTCN [2] proposes architecture for three types of performance testing: testing for a server, an end-to-end service and a protocol. It introduces foreground testers, which generate test load to the IUT; and background testers, which bring the IUT into a specific load state. PerfTTCN is an extension to TTCN-2. With the proposed new tables it is possible to describe the test configuration, traffic streams, traffic models, measurements, and to give performance constraints. Unfortunately, no known commercial TTCN-2 tool supports PerfTTCN.

Real-time TTCN [3] extends the capability of TTCN-2 to test real-time systems. It defines labels that specify the earliest and latest execution times for statements. It also defines an operational semantics for real-time TTCN by mapping it to timed transition systems.

TimedTTCN-3 [4] is a real-time extension to TTCN-3 that supports the test and measurement of real-time requirements. TimedTTCN-3 introduces absolute time, allows definition of synchronization requirements for test components and provides possibility to specify online and offline evaluation procedures for real-time requirements.

The authors of [5] showed a case study using TTCN-2 for performance test of an HTTP server. They applied the concepts of PerfTTCN, but not the language extensions. Thus, application of a commercial TTCN test executor was possible. The paper shows guidelines to design different parts of testing software (e.g. test port) that have to operate in a performance testing environment and consequently, must have themselves a good performance.

Some papers deal with investigation of BCMP. The most widely known contribution [6] presents the protocol itself and results of simulations performed with ns-2 simulation environment. The goal of simulations was to examine correctness and expected performance of BCMP.

The paper [7] describes a mobile testbed implementation using real network components. [8] concentrates on the scalability aspects of micro-mobility management from a theoretical point of view. Description of implementation of IP micro-mobility protocols and some tests can be found in [9]. Unfortunately, none of the papers above includes performance or scalability test results of networks running BCMP as mobility management protocol.

The main innovation of our measurements is the deployment of a TTCN-3 based parallel testing environment. This environment enables good approximation of real network scenarios while providing comparable results and being extremely flexible and reconfigurable.

3 Requirements on Performance Testing Environment Components

For a functional test of an implementation, in most cases, a few (or a single) test components are enough. However, performance tests may need several parallel components, e.g. foreground and background test components [2], monitor components etc. To control the start of test components a main control utility is needed. The components must have an internal communication protocol and path to communicate with each other and with the main control utility.

3.1 Distributed Parallel Test Environment

The architecture of our test system is based on the TTCN-3 standard [1]. Figure 1 depicts the distributed architecture and its components.

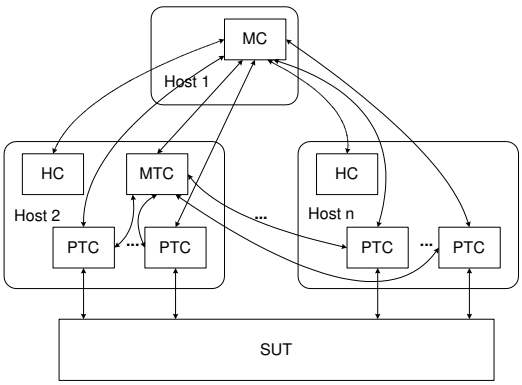

Fig. 1. Distributed test architecture

The *Main Controller (MC)* creates, starts and terminates the *Main Test Component (MTC)*. MC maintains a control connection with all other components. The *Host Controller (HC)* is responsible for controlling *Parallel Test Components (PTCs)* on a host. There is only one MTC in the whole distributed test system. MTC controls creation, start and termination of PTCs. MTC and PTCs are TTCN-3 test components while MC is a standalone utility.

Connections between different test components can carry protocol data used in testing and control messages between different test components. Connections between an abstract test component (either MTC or PTC) and the real test system interface are called mappings.

In a typical performance test configuration, several (possibly many) PTCs act as background testers, i.e., generate load against the *System Under Test (SUT)* and bring it into a specific load state. Some PTCs act as foreground testers and perform test sequences against the SUT. One of the advantages of TTCN-3

test environment is that backround testers are not implemented as dummy load generators but as test components that can fully track the protocol behaviour.

All of the test components are dynamically reconfigurable, thus enabling adaptation to different test scenarios.

3.2 Critical Performance Problems of the Test System

A performance test system in general, must have better performance than a test system used for functional test only. It has to keep correct timing, packet rate and response time. The test system hardware has to run its own operating system as well. It is essential to track the load on the test system and ensure that enough resources are available during the whole test campaign.

Performance Test Port The test port connects the abstract test system to the real test system interface. Since a test port is always implementation-dependent, it can not be coded in TTCN-3. One possibility is to write the test port in C++.

The code of the test port should remain small and should not use much CPU power. In many cases it is necessary to simplify an existing test port (that has been used for function testing) to be more efficient [5]. For example, many functions of the test port can possibly be unnecessary for background testers, but fast execution is a critical issue.

Data Definition Modules Data definition modules that describe *Abstract Service Primitives (ASPs)*, *Protocol Data Units (PDUs)* etc. of the tested protocols can be quite extensive, especially if the test system must cope with multiple protocols or protocol layers at the same time. Large size of modules implies long compilation time. Moreover, that is even more important, a large protocol definition module runs slowly on the test hardware.

Protocol data definitions given in ASN.1 are also problematic: in most cases, these modules are automatically compiled and the encoder/decoder functions are automatically generated. The resulting code has often a suboptimal performance. Consequently, data modules for performance test should be kept as small and simple as possible. One has to consider using manual encoding instead of ASN.1 automatic encoder/decoder functions.

Estimating Performance Limit of Test System When the load on SUT is increased, the load on test system also increases. If the load (processor or memory utilization, network bandwidth usage) on the test system is too high, it may degrade correct behaviour of the tester. A heavily loaded processor can not keep correct timing, consequently, it can not guarantee the required traffic and can not provide authentic test results.

It is essential to keep workload on test system below its upper bound. To achieve this, we need a method that indicates overloading of the test system.

4 BCMP Performance Test and Experiences

We chose a typical performance test configuration and process to show an example for the considerations mentioned above. In this section we describe the tested system, the objectives and tools of performance test. Finally, we present the results and our experiences.

4.1 System Under Test

The system that we investigated is an IP mobility test network. It consists of mobile IP network elements and test hosts. The network can work with several mobile IP protocols. We investigated *BRAIN Candidate Mobility Management Protocol (BCMP)* [6].

BCMP Network A BCMP network can be built on top of a legacy IP routed network, running an arbitrary routing protocol. Any number of IP routers may lie between network entities.

Anchor Routers (ANP) maintain a tunnel for each Mobile Node and transmit packets addressed to the Mobile Node toward its current location. ANPs are legacy routers with minimal BCMP specific functionality.

Access routers (AR) serve as attachment points for Mobile Nodes. They are routers at the edge of the network equipped with wireless interface. They manage handovers, maintain and transfer context for each served Mobile Node. ARs also act as packet filters and drop all packets that were not transmitted by/to authorized Mobile Nodes.

Mobile Nodes (MN) represent devices (subscriber's equipment) that wish to access the Internet via the network. They have a wireless interface and the appropriate protocol implementation.

Test Network The test network is depicted on Fig. 2.

We used three Access Routers (AR1, AR2, AR3) that connect to a single Anchor Router (ANP). Several hundred mobile nodes are necessary for testing the handover performance of the network. Moreover, these mobile nodes must be coordinated and controlled during the execution of tests. It is practically unfeasible to use real mobile terminals with wireless interface for this purpose. Instead, we emulated nodes in TTCN-3 *Mobile Node Emulator (MN Emulator)*. This tool is described in the following subsection.

The MN Emulator software runs on two hosts as a distributed parallel TTCN-3 test environment. The hosts are PCs with P-III 1 GHz processors and 256 MB of memory. We used Linux kernel version 2.4.20 on all PCs (with BCMP features developed at Ericsson). All connections are 100 Mbit Ethernet links.

Since the handover performance of the core BCMP netwok is independent of the physical layer, the wireless connections also can be substituted with Ethernet links.

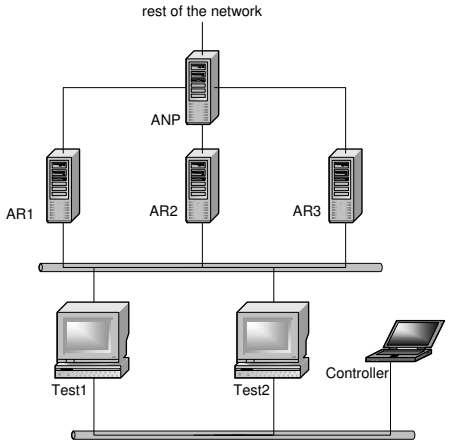

Fig. 2. BCMP test network

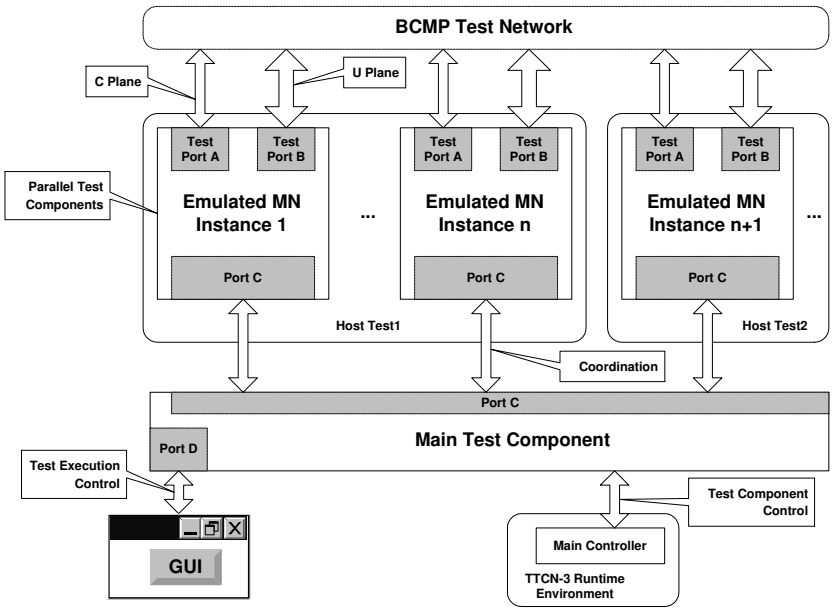

Fig. 3. Mobile Node Emulator – functional view

Mobile Node Emulator Figure 3 shows a functional view of MN Emulator.

The Mobile Nodes are represented by Parallel Test Components. The number of PTC instances can be set before the test execution. The behaviour of the mobility protocol is implemented in a protocol module. The tester can specify abstract test events (e.g. login, handover, logout) for each mobile node. Thus, it is possible to use an IP mobility protocol module other than BCMP without changing anything else in the test. This makes comparison of different IP mobility protocols easier.

Each Mobile Node has its own behaviour, message sequence and timer set theoretically independent from the others. However, in a real emulator, parallel test components running on a single-processor machine share the available common resources, e.g. processor and memory capacity. This leaves the problem of synchronization and can lead to undesirable interference that causes false test results (see Sect. 4.2).

Components of the test system communicate with themselves through internal message ports *(Port C, Port D)*. Communication towards the SUT is done through BCMP test port *(Test Port A, Test Port B)*. The BCMP test port is capable to work both with IPv4 and IPv6 network layer. The user plane traffic between mobile nodes is emulated as constant bitrate UDP traffic; i.e., UDP packets are sent out in equal time intervals. Although it is possible to implement a sophisticated traffic model in TTCN-3, in our case it would unacceptably degrade the performance of the test system due to the large number of test components.

The distribution of parallel test components on the two test hosts is done automatically by the TTCN-3 test environment.

The Graphical User Interface (GUI) is used to control execution of the test. In order to reduce load on the test system, it is run on a separate host.

Test Objectives The main goal of the test is to investigate handover transaction time (i.e., the length of time interval between the beginning and the successful end of a handover) as a function of varying handoff activity of Mobile Nodes. The handover transaction time influences the number of lost packets during a handover event, thus the quite short handover transaction time is critical from the point of view of a real-time application.

We aimed to produce performance test results that are comparable to other BCMP implementations and different IP micro-mobility protocols. Our tests verify that the BCMP network can scale to handle several hundreds of handover events within a short time. In the following scenario we used 300 emulated Mobile Nodes performing periodic handovers. We measured the processor load on ARs, handover transaction time and message load.

4.2 Test Results

On Fig. 4 the average AR processor load is shown.

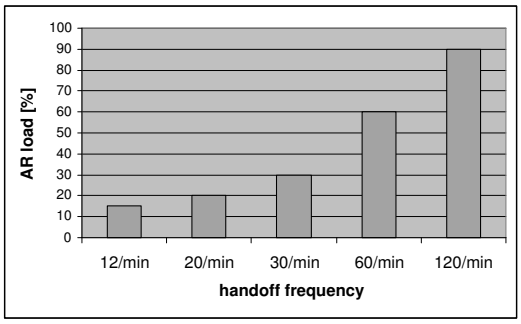

Fig. 4. Access router processor load

Figure 5 shows the average length of a handover (as seen by the MN). It can be seen that below a critical point, increasing number of handovers results in gracefully degrading handover performance (i.e., increasing handover transaction time). Going higher with handover frequency, the load on the AR reaches a critical value resulting in rapidly increasing handover transaction times, which results in very poor handover performance. A preliminary conclusion is that the system sholud be dimensioned such that ARs stay below this critical point.

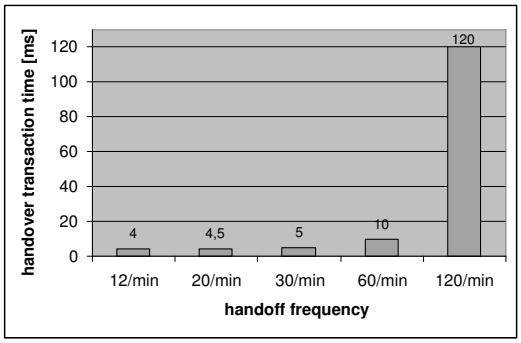

Fig. 5. Handover transaction time

Figure 6 depicts the measured average BCMP message load of the handover activity.

Experiences During the development of MN Emulator we used some practical tricks that enabled emulation of many nodes with keeping the load on the test system below its upper bound.

If the load on the test host is near to the full utilization of resources (processor and memory usage) then it can be obviously observable through quick decrease of

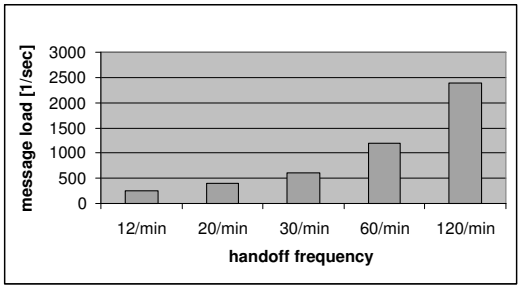

Fig. 6. Message load

processor idle time and available memory, respectively. We used an other method for observing the reach of load limit of test system. In the log files the actual timestamps of initiated test events are registrated. If these timestamps deviates significantly from the specified event times (a slip comes in the execution) then the system is near to its upper bound of performance and it results incorrect test operation.

It happens often in the test that several MNs have to perform an event (e.g. a handover) at the same time. Obviously, in a one-processor system these events occur not exactly in the same point of time. If we still specify the execution for the same time, it results sharp load-peaks or even an overload for the test system. For that reason timing of events to be generated by Mobile Nodes is randomized. A $\Delta\tau$ offset of constant distribution stochastic variable is added to the mean execution time. This way the load on test system can be smoothed.

We observed that sometimes it is not efficient to extensively use the `alt` mechanism of TTCN-3 in the performance-critical main event handler loop of parallel test components. This property comes from the standard TTCN-3 semantics and does not depend on the actual realization of the test execution environment. Let us consider the following example:

```
alt
{
    [] MyPort.receive(template_1) {...}
    [] MyPort.receive(template_2) {...}
    ...
    [] MyPort.receive(template_n) {...}
}
```

When the execution of the test is at an `alt` statement and a message is arrived, in a worst case situation, all n templates must be compared to the message. If a few types of messages are expected to arrive significantly more often than other ones, then it is better to use fewer, more general templates and analyse the incoming message with `if()` conditional operators inside the `alt` construct. Alternatively, use of multiple levels of `alt` statements with less alternatives in the same level also solves the problem. Obviously, the most frequent message

should stand on the first place within an `alt`. We can thus save the time of some template matching operations.

It is necessary to use templates as simple as possible for template matching. Outgoing messages should not contain complicated structures, because they slow down coding operations and result huge binary executables. Sometimes this can lead to a difficult design problem.

With the considerations above we achieved that 600 emulated Mobile Nodes can run on the two test hosts without overloading the test system.

5 Conclusions

In this paper we presented a BCMP performance test example focusing on some problems of optimization of the test system. We described the distributed parallel TTCN-3 test system, the investigated test network and the Mobile Node Emulator. Performance test optimization techniques and two methods of observing load limit of the test system were also presented. We showed some results of the measurements and summarized our most important experiences.

Regarding further inverstigations, it would be interesting to study the possibility of using behaviour and traffic models in the emulator instead of simple periodic event generation. Thus, a real scenario could be more precisely approximated. On the other hand, a complicated traffic model results worse performance for the test system. We plan to find a compromise among these requirements.

References

1. ETSI Methods for Testing and Specification, The Testing and Test Control Notation version 3, Part 1: TTCN-3 Core Language. ETSI ES 201 873-1 V2.2.0 (2002-03)
2. Ina Schieferdecker, Bernard Stepien, Axel Rennoch: PerfTTCN, a TTCN Language Extension for Performance Testing. IWTCS 1997, Cheju Island, Korea. Testing of Communicating Systems volume 10, Chapman & Hall.
3. Thomas Walter, Jens Grabowski: Real-Time TTCN for Testing Real-Time and Multimedia Systems. IWTCS 1997, Cheju Island, Korea. Testing of Communicating Systems volume 10, Chapman & Hall.
4. Zhen Ru Dai, Jens Grabowski, Helmut Neukirchen: TimedTTCN-3 – A Real-Time Extension for TTCN-3. The IFIP 14th International Conference on Testing of Communicating Systems, 2002.
5. Roland Gecse, Péter Krémer, János Zoltán Szabó: HTTP Performance Evaluation with TTCN. The IFIP 13th International Conference on Testing of Communicating Systems, Canada, 2000.
6. C. Keszei, N. Georganopoulos, Z. Turányi, A. Valkó: Evaluation of the BRAIN Candidate Mobility Management Protocol. IST Global Summit 2001, Barcelona, September 2001.
7. J.C. Rault, L. Burness, E. Garca, T. Robles, J. Manner, N. Georganopoulos, P. Ruiz: IP QoS and mobility experimentations within the MIND trial Workpackage. PIMRC2002, Lisbon, Portugal, September 2002 (http://www.lx.it.pt/pimrc2002/)

8. P. Eardley, N. Georganopoulos, M. West: On the Scalability of IP micro mobility management protocols. IEEE Conference on Mobile and Wireless Communication Networks (MCWN2002) Stockholm, Sweden, 9-11 September 2002
9. K. Guillouard, Y. Khouaja, J.C. Rault: Advanced IP mobility management experimentation within a Wireless Access Network WTC2002 Paris, 26th September 2002