SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE

NOTICE

While reasonable efforts have been made to assure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be reliable. However, no responsibility is assumed for inaccuracies or omissions. Telit reserves the right to make changes to any products described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. Telit does not assume any liability arising out of the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.

It is possible that this publication may contain references to, or information about Telit products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Telit intends to announce such Telit products, programming, or services in your country.

COPYRIGHTS

This instruction manual and the Telit products described in this instruction manual may be, include or describe copyrighted Telit material, such as computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any copyrighted material of Telit and its licensors contained herein or in the Telit products described in this instruction manual may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of Telit. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit, as arises by operation of law in the sale of a product.

COMPUTER SOFTWARE COPYRIGHTS

The Telit and 3rd Party supplied Software (SW) products described in this instruction manual may include copyrighted Telit and other 3rd Party supplied computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and other 3rd Party supplied SW certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted Telit or other 3rd Party supplied SW computer programs contained in the Telit products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of Telit or the 3rd Party SW supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit or other 3rd Party supplied SW, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.
USAGE AND DISCLOSURE RESTRICTIONS

I. License Agreements

The software described in this document is the property of Telit and its licensors. It is furnished by express license agreement only and may be used only in accordance with the terms of such an agreement.

II. Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of Telit.

III. High Risk Materials

Components, units, or third-party products used in the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: the operation of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems (High Risk Activities*). Telit and its supplier(s) specifically disclaim any expressed or implied warranty of fitness for such High Risk Activities.

IV. Trademarks

TELIT and the Stylized T Logo are registered in Trademark Office. All other product or service names are the property of their respective owners.

V. Third Party Rights

The software may include Third Party Right software. In this case you agree to comply with all terms and conditions imposed on you in respect of such separate software. In addition to Third Party Terms, the disclaimer of warranty and limitation of liability provisions in this License shall apply to the Third Party Right software.

TELIT HEREBY DISCLAIMS ANY AND ALL WARRANTIES EXPRESS OR IMPLIED FROM ANY THIRD PARTIES REGARDING ANY SEPARATE FILES, ANY THIRD PARTY MATERIALS INCLUDED IN THE SOFTWARE, ANY THIRD PARTY MATERIALS FROM WHICH THE SOFTWARE IS DERIVED (COLLECTIVELY “OTHER CODE”), AND THE USE OF ANY OR ALL THE OTHER CODE IN CONNECTION WITH THE SOFTWARE, INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF SATISFACTORY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE.

NO THIRD PARTY LICENSORS OF OTHER CODE SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND WHETHER MADE UNDER CONTRACT, TORT OR OTHER LEGAL THEORY, ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE OTHER CODE OR THE EXERCISE OF ANY RIGHTS GRANTED UNDER EITHER OR BOTH THIS LICENSE AND THE LEGAL TERMS APPLICABLE TO ANY SEPARATE FILES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
**APPLICABILITY TABLE**

<table>
<thead>
<tr>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GS2K based Modules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SW Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.x onwards</td>
</tr>
</tbody>
</table>

**Note:** The features described in the present document are provided by the products equipped with the software versions equal or higher than the versions shown in the table. See also the Revision History chapter.
### Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>August 2015</td>
<td>Initial draft</td>
</tr>
<tr>
<td>2.0</td>
<td>July 2016</td>
<td>Added <a href="#">Chapter 4. Measure Throughput in RF Chamber</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated <a href="#">Chapter 5. Reference Code</a></td>
</tr>
<tr>
<td>3.0</td>
<td>November 2017</td>
<td>Added section <a href="#">4.1 Test Setup</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified section heading <a href="#">4.2 Initial Setup</a> to <a href="#">4.2 Initial Setup for UDP and TCP</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the following figures related to generating S2W throughput binaries</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="#">Figure 1: Select S2W Project File</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="#">Figure 2: Select Batch File</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="#">Figure 3: Running GS Module Program Script</a></td>
</tr>
<tr>
<td>3.1</td>
<td>May 2018</td>
<td>Added a note about IAR IDE Installation in <a href="#">Chapter 2 Generate Throughput Binary</a></td>
</tr>
<tr>
<td>3.2</td>
<td>September 2018</td>
<td>Updated section <a href="#">3.2.1.1 GS2K Limited-AP Client Tx</a></td>
</tr>
</tbody>
</table>
# Table of Contents

NOTICE .......................................................................................................................... 2
COPYRIGHTS ................................................................................................................... 2
COMPUTER SOFTWARE COPYRIGHTS ........................................................................ 2
USAGE AND DISCLOSURE RESTRICTIONS ................................................................. 3
APPLICABILITY TABLE .................................................................................................. 4

CHAPTER 1. INTRODUCTION ......................................................................................... 9
  1.1 PURPOSE .................................................................................................................. 9
  1.2 SCOPE ...................................................................................................................... 9
  1.3 TERMINOLOGY ........................................................................................................ 9

CHAPTER 2. GENERATE THROUGHPUT BINARY ...................................................... 10

CHAPTER 3. MEASURE THROUGHPUT ....................................................................... 13
  3.1 PREREQUISITES ...................................................................................................... 13
  3.2 TCP ......................................................................................................................... 13
    3.2.1 Limited-AP Mode .............................................................................................. 13
    3.2.2 STA Mode ......................................................................................................... 17
  3.3 UDP ......................................................................................................................... 23
    3.3.1 Limited-AP Mode .............................................................................................. 23
    3.3.2 STA Mode ......................................................................................................... 25

CHAPTER 4. MEASURE THROUGHPUT IN RF CHAMBER ........................................... 28
  4.1 TEST SETUP ............................................................................................................ 28
  4.2 INITIAL SETUP FOR UDP AND TCP ................................................................. 29
    4.2.1 Prerequisites ..................................................................................................... 29
  4.3 CONFIGURATION SETUP ...................................................................................... 29

CHAPTER 5. REFERENCE CODE .................................................................................. 30
  5.1 THROUGHPUT TEST BASED ON NUMBER OF PACKETS .................................... 30
  5.2 THROUGHPUT TEST BASED ON TIME ................................................................. 41
List of Figures

Figure 1: Select S2W Project File .......................................................... 10
Figure 2: Select Batch File ................................................................. 11
Figure 3: Running GS Module Program Script ........................................ 11
Figure 4: Transmission between TCP Client and TCP Server ..................... 13
Figure 5: Reception between TCP Server and TCP Client .......................... 15
Figure 6: Transmission between TCP Client and TCP Server ..................... 18
Figure 7: Reception between TCP Server and TCP Client .......................... 20
Figure 8: Transmission between UDP Client & UDP Server ....................... 23
Figure 9: Reception between UDP Server & UDP Client ........................... 24
Figure 10: Transmission between UDP Client & UDP Server ..................... 25
Figure 11: Reception between UDP Server & UDP Client .......................... 26
Figure 12: Test Setup Connection inside RF Chamber ............................... 28
Figure 13: Throughput Test Setup for GS2K module ................................. 29
List of Tables

Table 1: Glossary of Terms ........................................................................................................... 9
Chapter 1. Introduction

1.1 Purpose
This document helps to measure throughput of GS2K module using TCP and UDP protocols. Throughput measurement of GS2K module is performed in Limited-AP and STA modes. It can be used for system throughput measurements not involving host interface, throughput over range measurements, radio performance measurements, and so on. This path exercises TCP/IP network stack, socket interface, APP-WLAN shared memory interface, WLAN MAC firmware, and WLAN MAC/PHY/RF hardware. This method does not involve host interface (host interface with external MCU is not involved) and therefore not suitable for those measurements.

1.2 Scope
This document provides information about throughput measurement of GS2K module using TCP and UDP protocols. For TCP and UDP throughput measurement, use Windows 7/XP PC with Cygwin.

1.3 Terminology
The following table lists the terminology used in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>Limited-AP</td>
<td>Limited AP mode</td>
</tr>
<tr>
<td>STA</td>
<td>Station or Client mode</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>DUT</td>
<td>Device Under Test</td>
</tr>
</tbody>
</table>
Chapter 2. Generate Throughput Binary

Perform the following steps to build and program S2W binaries for testing throughput:

1. Login to www.gainspan.com and download the latest SDK package.
2. Locate the S2W project file (s2w.eww) from userapps\s2w\build.

**NOTE** – For “IAR IDE” installation please refer “GS2K SDK Application Reference Guide” – section “Installing IAR IDE” of Chapter 2 - IAR.

3. Open the project file in IAR and rebuild the code by enabling the macro S2W_THROUGHPUT_TEST.
4. After successful compilation run, locate the batch file flash_program_FlashFetch_SingleImage.bat placed in the same path and follow the steps as guided on terminal.
NOTE:
Sample program of TCP is compatible for compilation and running on Cygwin under windows 7 platform.

5. Enter the com port when indicated.
6. Single image is created and the board is directed to program mode and power cycle is performed.
7. The board is now successfully erased and flashed with the image.
8. Once the board is programmed, change it to run mode and reboot the board.
Chapter 3. Measure Throughput

3.1 PREREQUISITES

Setup required to measure throughput in Limited-AP and STA modes:

- GS2K Evaluation Board
- PC with TCP/UDP Socket applications
- GS2K S2W Single image with throughput binary (Refer Chapter 2. Generate Throughput Binary)
- TCP/UDP test application for PC (provided with package)
- USB cables for TeraTerm interface
- Third party standard b/g/n AP (Used only in STA mode)

3.2 TCP

TCP protocol is used for reliable communication as it verifies end-to-end connection. It is the backbone of many application level protocols such as HTTP, RTSP, and so on. The following sections provide different setup’s to measure throughput in Limited-AP and STA modes using TCP protocol.

3.2.1 Limited-AP Mode

3.2.1.1 GS2K Limited-AP Client Tx

Setup Diagram:

Procedure:

Procedure for TCP Client Tx using GS2K as Limited-AP is as follows:

1. Boot up GS2K module with S2W application
2. Create a Limited-AP
   
   AT+NSET=192.168.40.1, 255.255.255.0, 192.168.40.1 //Configure IP gateway.
   
   AT+DHCPSRV=1 //Start DHCP Server.
   
   AT+WM=2 // Configure GS2K as Limited-AP.
   
   AT+WA=<SSID>,,1 //Create Limited-AP with given SSID.

3. Connect PC to GS2K Limited-AP
4. Start TCP Server on PC
   
   ./throughput.exe
5. Next, run the TCP/UDP server on your PC by issuing the following command

   `$ ./throughput_org.exe <TCP Server> <Default port number>`

   Example: `$ ./throughput.exe 3 8000`

6. Create TCP Client on GS2K module

   `AT+NCTCP=192.168.40.x, <port number>`

7. Issue throughput command

   `AT+SOCKTPTEST=<CID>, <Mode>, <Iteration>, <Packet length>`

   Example: `AT+SOCKTPTEST=0,0,1000,1400`

8. Wait for 5 seconds after transmission is complete on GS2K for PC TCP Server to print the throughput statistics

3.2.1.2 GS2K Limited-AP Server Rx

Setup Diagram:
**Procedure:**

Procedure for TCP Server Rx using GS2K as Limited-AP is as follows:

1. Boot up GS2K module with S2W application.
2. Create a Limited-AP.
   
   AT+NSET=192.168.40.1, 255.255.255.0, 192.168.40.1 //Configure IP gateway.
   
   AT+DHCP=1 //Start DHCP Server.
   
   AT+WM=2 // Configure GS2K as Limited-AP.
   
   AT+WA=<SSID>,,1 //Create Limited-AP with given SSID.
3. Connect PC to GS2K Limited-AP.
4. Create TCP Server on GS2K module.
   
   AT+NSTCP=<port number>
5. Start TCP Client on PC.
   
   $ ./throughput.exe <IP of GS2K module><SP><port number><SP><No. of packets to be received><SP><Size of each packet>

Example: $ ./throughput.exe 192.168.40.1 8000 1000 1400

6. GS2K TCP Server receives a connect notification from TCP Client on PC.
7. Issue throughput command.
   \texttt{AT+SOCKTPTEST=<client cid>,4}

   \textbf{Example:} \texttt{AT+SOCKTPTEST=1,4}

8. Enter \textbf{any} number and press \textbf{Enter} key to start data Tx from client Application.

9. Wait for data Tx complete message from client application.
10. After receiving the message on Application terminal, issue throughput statistics command.

   AT+SOCKTPTESTSTAT =<cid>

   Example: AT+SOCKTPTESTSTAT =1

### 3.2.2 STA Mode

The test cases for STA mode remain same as the Limited-AP mode except that both GS node and Laptop are STA, and communicate with each other through a standard AP.

#### 3.2.2.1 GS2K STA Client Tx

**Setup Diagram:**
Procedure:

Procedure for TCP Client Tx using GS2K as STA is as follows:

1. Boot up GS2K module with S2W application.
2. Connect GS2K to an AP.
   
   $ AT+WRXACTIVE=1 //Set receiver to Always ON.
   $ AT+WRXPS=0 //Disable PS-Poll.
   $ AT+NDHCP=1 //Enable DHCP Client.
   $ AT+WA=<SSID>,,1 //Connect to an AP with SSID.

3. Connect PC to the same AP.
4. Start TCP Server on PC.
   
   $ ./throughput.exe 3 <port number>

5. Create TCP Client on GS2K module.
   
   $ AT+NCTCP=192.168.40.x,<port number>

6. Issue throughput command.
   
   $ AT+SOCKTPTTEST=<CID>,<Mode>,<Iteration>,<Packet length>
   
   Example: AT+SOCKTPTTEST=0,0,1000,1400

7. Wait for 5 seconds after transmission is complete on GS2K for PC TCP Server to print the throughput statistics.
### 3.2.2.2 GS2K STA Server Rx

#### Setup Diagram:

![Diagram showing GS2K STA Server Rx setup](image)

**Figure 7: Reception between TCP Server and TCP Client**

#### Procedure:

Procedure for TCP Server Rx using GS2K as STA is as follows:

1. Boot up GS2K module with S2W application.
2. Connect GS2K to a third-party AP.
   
   ```
   AT+WRXACTIVE=1 //Set receiver to Always ON.
   AT+WRXPS=0  //Disable PS-Poll.
   AT+NDHCP=1 //Enable DHCP Client.
   AT+WA=<SSID>,,1 //Connect to an AP with provided SSID.
   ```
3. Connect PC to the same AP.
4. Create TCP Server on GS2K module.
   
   ```
   AT+NSTCP=<port number>
   ```
5. Start TCP Client on PC.
   
   ```
   $ ./throughput.exe 2 <IP of GS2K module><SP><port number><SP><No. of packets to be received><SP><Size of each packet>
   ```
   
   Example: $ ./throughput.exe 2 192.168.40.1 8000 1000 1400
6. GS2K TCP Server receives a connect notification from TCP Client on PC.

7. Issue throughput command.

   \texttt{AT+SOCKPTEST=<client cid>,4}

   \textbf{Example:} \texttt{AT+SOCKPTEST=1,4}
8. Enter any number followed with **Enter** key to start Data TX from client Application.
9. Wait for data Tx complete message from client application.
10. After receiving the message on Application terminal, issue throughput statistics command.

   AT+SOCKTPTESTSTAT =<cid>

   **Example:** AT+SOCKTPTESTSTAT =1

3.3 **UDP**

UDP is used to independently transmit and receive throughput measurement as there is only one-way traffic. The following sections provide different setups to measure throughput in Limited-AP and STA modes using UDP protocol.

### 3.3.1 **Limited-AP Mode**

#### 3.3.1.1 **GS2K Limited-AP Client Tx**

**Setup Diagram:**

![Figure 8: Transmission between UDP Client & UDP Server](image)

**Procedure:**

Procedure for UDP Client Tx using GS2K as Limited-AP is as follows:

1. Boot up GS2K module with S2W application.
2. Create a Limited-AP

   AT+NSET=192.168.40.1, 255.255.255.0, 192.168.40.1 //Configure IP gateway.

   AT+DHCP=1 //Start DHCP Server.
AT+WM=2 // Configure GS2K as Limited-AP.
AT+WA=<SSID>,,1 // Create Limited-AP with given SSID

3. Connect PC to GS2K Limited-AP.
4. Start UDP Server on PC.
   $ ./throughput.exe 1 <port number>

5. Create UDP Client on GS2K module.
   AT+NCUDP=192.168.40.x,<port number>

6. Issue throughput command.
   AT+SOCKTPTEST=<CID>,<Mode>,<Iteration>,<Packet length>
   Example: AT+SOCKTPTEST=0.0,1000,1400

7. Wait for 5 seconds after transmission is complete on GS2K for PC UDP Server to print the throughput statistics.

### 3.3.1.2 GS2K Limited-AP Server Rx

**Setup Diagram:**

![Figure 9: Reception between UDP Server & UDP Client](image)

**Procedure:**

Procedure for UDP Server Tx using GS2K as Limited-AP is as follows:

1. Boot up GS2K module with S2W application.
2. Create a Limited-AP.
   
   AT+NSET=192.168.40.1, 255.255.255.0, 192.168.40.1 // Configure IP gateway.
   
   AT+DHCPSRVR=1 // Start DHCP Server.
   
   AT+WM=2 // Configure GS2K as Limited-AP.
   
   AT+WA=<SSID>,,1 // Create Limited-AP with given SSID.

3. Connect PC to GS2K Limited-AP.
4. Create UDP Server on GS2K module.
   
   AT+NSUDP=<port number>

5. Start UDP Client on PC.
   
   $ ./throughput.exe 0 <IP of GS2K module><SP><port number><SP><No. of packets to be received><SP><Size of each packet>
   
   Example: $ ./throughput.exe 0 192.168.40.1 8000 1000 1400
6. Issue throughput command.
   \[ \text{AT+SOCKTPTEST} = \langle \text{client cid} \rangle, 4 \]
   \textbf{Example:} AT+SOCKTPTEST=1,4

7. Wait for data Tx complete message from client application.
8. After receiving the message on Application terminal, issue throughput statistics command.
   \[ \text{AT+SOCKTPTESTSTAT} = \langle \text{cid} \rangle \]
   \textbf{Example:} AT+SOCKTPTESTSTAT =

3.3.2 STA Mode
3.3.2.1 GS2K STA Client Tx

\textbf{Setup Diagram:}

\begin{center}
\includegraphics[width=\textwidth]{figure10.png}
\end{center}

\textbf{Figure 10: Transmission between UDP Client & UDP Server}

\textbf{Procedure:}

Procedure for UDP Client Tx using GS2K as STA is as follows:

1. Boot up GS2K module with S2W application.
2. Connect GS2K to an AP.
   \[ \text{AT+WRXACTIVE=1} \] //Set receiver to Always ON.
   \[ \text{AT+WRXPS=0} \] //Disable PS-Poll.
   \[ \text{AT+NDHCP=1} \] //Enable DHCP Client.
   \[ \text{AT+WA=<SSID>,1} \] //Connect to an AP with SSID.

3. Connect PC to the same AP.
4. Start UDP Server on PC.
$ ./throughput.exe 1 <port number>

5. Create UDP Client on GS2K module.
   AT+NCUDP=192.168.40.x,<port number>

6. Issue throughput command.
   AT+SOCKTPTEST=<CID>,<Mode>,<Iteration>,<Packet length>
   Example: AT+SOCKTPTEST=0.0,1000,1400

7. Wait for 5 seconds after transmission is complete on GS2K for PC UDP Server to print the throughput statistics.

3.3.2.2 GS2K STA Server Rx

Setup Diagram:

![Setup Diagram](image)

**Figure 11: Reception between UDP Server & UDP Client**

Procedure:

Procedure for UDP Server Tx using GS2K as STA is as follows:

1. Boot up GS2K module with S2W application.
2. Connect GS2K to a third-party AP.
   
   AT+WRXACTIVE=1 //Set receiver to Always ON.
   AT+WRXPS=0     //Disable PS-Poll.
   AT+NDHCP=1     //Enable DHCP Client.
   AT+WA=<SSID>,,1 //Connect to an AP with provided SSID.

3. Connect PC to the same AP.
4. Create UDP Server on GS2K module.
   AT+NSUDP=<port number>
5. Start UDP Client on PC.
   $ ./throughput.exe 0 <IP of GS2K module><SP><port number><SP><No. of packets to be received><SP><Size of each packet>
   Example: $ throughput.exe 0 192.168.40.1 8000 1000 1400

6. Issue throughput command.
   AT+SOCKTPTEST=<client cid>,4
   Example: AT+SOCKTPTEST=1,4

7. Wait for data Tx complete message from client application.
8. After receiving the message on Application terminal, issue throughput statistics command.
   AT+SOCKTPTESTSTAT =<cid>
   Example: AT+SOCKTPTESTSTAT =1
Chapter 4. Measure Throughput in RF Chamber

The RF Chamber provides a controlled RF environment with stable and repeatable results which is difficult to obtain in an open-air condition due to changing channel conditions such as time, frequency and position.

4.1 Test Setup

The following figure demonstrates a typical RF chamber test setup for measurement of throughput performance.

![Figure 12: Test Setup Connection inside RF Chamber](image)

The procedure for the setup is as follows:

In this setup, the GS node acts as a client and the PC acts as a server.

1. Connect the USB hub and GS2K modules as shown in [Figure 12](image).
2. Once the setup is done, close the RF chamber lid and run the AT commands from your PC. Refer to section Chapter 3. Measure Throughput for AT commands.
4.2 **INITIAL SETUP FOR UDP AND TCP**

4.2.1 **Prerequisites**

1. TeraTerm or any such software for sending and receiving Serial port data.
2. UDP and TCP Socket Applications. Install Cygwin on PC to run these applications.

The applications reference a couple of the dll files which are compiled for Windows. The following diagram illustrates the setup for UDP and TCP throughput tests in RF Chamber.

![Throughput Test Setup for GS2K module](image)

The initial setup to measure throughput in RF chamber is as follows:

1. The DUT and Cisco AP (or any third-party AP) are placed in two separate RF shielded chambers and are connected to each other across an attenuator using SMA cables and SMA to UFl adapters where needed. This setup provides an ideal medium of communication between the devices.
2. The PC communicates with GS2K module using a UART cable which is also wired into the shield box.
3. The PC communicates with the AP via LAN cable. The LAN cable is used to send and receive data as well as to configure the AP parameters such as SSID, Channel, and so on.

4.3 **CONFIGURATION SETUP**

Perform the following steps to configure the setup for throughput measurement in RF Chamber:

1. Access the AP’s html page by entering its IP address in the browser. Configure it to require channel and set other parameters such as SSID, RF operational mode, auto data rate or fixed rate, and so on. For a basic test, use Auto rate to allow the AP to use rate selection.
2. Set the value needed on the attenuator. Make sure there is a path loss of at least 40 dB in the beginning so that the signal between the two devices is not too strong.
3. Program the module with the desired firmware and set it to run mode.
4. Make sure calibration for the module is completed for all buckets by raising the temperature and running the AT+WLCALSTART=1 command in a loop during both the heating phase and the cooling phase.
5. Once the setup is done, perform the steps provided in Chapter 5. Reference Code.
6. Measure Throughput as applicable.
Chapter 5. Reference Code

5.1 Throughput Test Based on Number of Packets

The following is a reference code for TCP/UDP Client and Server applications:

```c
#include "windows.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <netdb.h>
#include <time.h>
#include <math.h>
#include "memory.h"
#define DEFAULT_PORT 8000
#define MYPORT "8000" // Default port number users will be connecting to
#define TIMEOUT_PERIOD_IN_SEC 5 /* Time out after the last packet reception */
#define TX_PACKET_MAX_SIZE 1496

char portNum_udpserver[8];
#include "memory.h"

// DEBUG_ENABLE

unsigned int fill_tx_data(char *data, unsigned int max_size) {
    unsigned int i = 0;
    for (i = 0; i < max_size; i++) {
        *(data+i) = (unsigned char)i;
    }
    return max_size;
}

unsigned int udp_client(const char *ipaddress, unsigned short port,
                         unsigned int numofpacket, unsigned int packetsize,
                         unsigned int totalbytes) {
    unsigned int retval = 0;
    int sockfd;
    struct sockaddr_in servaddr;
    char sendline[TX_PACKET_MAX_SIZE+1];

    #ifdef DEBUG_ENABLE
    printf("\r\n UDP Client Called ");
    #endif

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr=inet_addr(ipaddress);
servaddr.sin_port=htons(port);

fill_tx_data(sendline,packetsize);

printf("\r\n\n**************$Send Statistics Start \n**************\r\n");

printf("\r\nUDP Client IP: %s. Port Num:%d \r\n",ipaddress, port);
printf("\r\n\nTotal Tx packets:%lld . Size :%lld \r\n",numofpacket,packetsize);

while (numofpacket > 0)
{
    sendto(sockfd,sendline,packetsize,0,
            (struct sockaddr *)&servaddr,sizeof(servaddr));

    numofpacket--;
}

close(sockfd);

printf("\r\n**************$Send Statistics End \n**************\r\n");
return retVal;
}

char buf[TX_PACKET_MAX_SIZE+1];
char s[INET6_ADDRSTRLEN];

unsigned int udp_server(const char* ipaddress, unsigned short port,
            unsigned int numofpacket, unsigned packetsize,
            unsigned int totalbytes)
{
    unsigned int retVal=0;
    struct timeval tv;
    int sockfd;
    int recvflag = 1;
    struct sockaddr_in their_addr;
    struct sockaddr_in servaddr,cliaddr;
    socklen_t addr_len;
    FILETIME sysTime;
    long long startTime=0;
    long long endTime=0;
    long long timeDurationInMicSec;
    volatile long total_no_of_packets=0;
    long expected_length=0;
    volatile int numbytes=0;
    volatile long long length = 0;
    double totalTxBits=0;
    double dataRate=0;
    fd_set rfds;
    int retval;

    struct addrinfo hints, *servinfo, *p;
    int rv;

#elif DEBUG_ENABLE
printf("\r\n UDP Server Called \r\n");
#endif

tv.tv_sec = TIMEOUT_PERIOD_IN_SEC; /* Time out after the last packet reception */
tv.tv_usec = 0;

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    perror("listener: socket");
    exit(1);
}

bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(port);

if (bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr)) == -1)
{
    close(sockfd);
    perror("listener: bind");
    exit(1);
}

printf("UDP Server started on port: %d\r\n",port);
total_no_of_packets=0;

while(recvflag)
{

    addr_len = sizeof (cliaddr);

    if ((numbytes = recvfrom(sockfd, buf, 1400, 0, (struct sockaddr *)&cliaddr, &addr_len)) < 0)
    {
        printf("\r\n Recv Failed \r\n");
        perror("recvfrom");
        //exit(1);
    }

    total_no_of_packets++;

    #ifdef DEBUG_ENABLE
    printf("listener: got packet from %s\n",inet_ntop(cliaddr.sin_family, get_in_addr((struct sockaddr *)&cliaddr), s, sizeof(s)));
    #endif /* DEBUG_ENABLE */

    if (0==length)
    {
        GetSystemTimeAsFileTime(&sysTime);
        startTime = sysTime.dwHighDateTime;
        startTime = (startTime<<32) | sysTime.dwLowDateTime; /* number of 100-nanosecond intervals */
    }

    length+=numbytes;
buf[numbytes] = '\0';
FD_ZERO(&rfds);
FD_SET(sockfd, &rfds);
retval = select(sockfd+1, &rfds, NULL, NULL, &tv);

if (retval <= 0)
{
    GetSystemTimeAsFileTime(&sysTime);
    endTime = sysTime.dwHighDateTime;
    endTime = (endTime<<32) | sysTime.dwLowDateTime; /* number of 100-nanosecond intervals */
    timeDurationInMicSec = (endTime - startTime) / (1ull*10); /* Subtract the last time-out value */
    if(timeDurationInMicSec)
    {
        timeDurationInMicSec -= ((1ull*1000*1000));
    }
    printf("$Receive Statistics Start
***************$Receive Statistics Start
$Time taken: %ld micro sec
",timeDurationInMicSec);
    printf("$Convert to bits /
totalTxBits = length*8;
    if(totalTxBits)
    {
        dataRate = (totalTxBits*1000*1000)/timeDurationInMicSec;
    }
    if(dataRate)
    {
        printf("$Number of packets received
: %lld
",total_no_of_packets));
        printf("$Total number of bytes received
: %lld
",length));
        if(0 != expected_length)
        {
            if(expected_length == length)
            {
                printf("$Data Rate: %f kbps",(dataRate/1024));
            }
        }
        printf("$Data Rate: %f mbps",(dataRate/(1024*1024)));
    }
    else
    {
        printf("$Error: Unable to calculate throughput.");
    }
unsigned int tcp_client(const char* ipaddress, unsigned short port, 
   unsigned int numofpacket, unsigned packetsize, 
   unsigned int totalbytes)
{
    unsigned int retVal=0;
    int sockfd, num;
    struct sockaddr_in servaddr;
    char sendline[TX_PACKET_MAX_SIZE+1];

    printf("\nTCP Client Called ");
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=inet_addr(ipaddress);
    servaddr.sin_port=htons(port);
    fill_tx_data(sendline,packetsize);

    if(connect(sockfd,(struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        printf(" Connect Error \n");
        close(sockfd);
        return 0;
    }

    printf("\nExecute at+socktptest command on DUT now and then Enter a 
number to continue\n");
    scanf("%d", &num);

    printf("\nTCP Client IP: %s. Port Num:%d ");
    printf("\nTotal Tx packets:%lld . Size :%lld ");

    while (numofpacket > 0)
    {
        send(sockfd,sendline,packetsize,0);
        numofpacket--;
    }
}  
    printf("Check throughput on DUT now and then Enter a number to continue\n");
    scanf("%d", &num);
    close(sockfd);

    printf("End of Transmission here!\n");

    return retVal;
}

unsigned int tcp_server(const char* ipAddress, unsigned short port, unsigned int numOfPacket, unsigned int packetSize, unsigned int totalBytes) {
    unsigned int retVal = 0;
    struct timeval tv;
    int sockfd, datasockFd;
    struct sockaddr_in server_addr, client_addr;
    int clientLen;
    int acceptflag = 1, recvflag = 1;
    struct sockaddr_storage their_addr;
    socklen_t addr_len;
    FILETIME sysTime;
    long long startTime = 0;
    long long endTime = 0;
    long long timeDurationInMicSec;
    volatile long total_no_of_packets = 0;
    long expected_length = 0;
    volatile int numbytes = 0;
    double totalTxBits = 0;
    double dataRate = 0;
    fd_set rfds;
    int retval;
    char buf[TX_PACKET_MAX_SIZE+1];

    printf("TCP Server Called ");

    tv.tv_sec = TIMEOUT_PERIOD_IN_SEC; /* Time out after the last packet reception */
    tv.tv_usec = 0;

    if ((sockFd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket Error");
        exit(1);
    }  

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
        perror("Error: Unable to bind");
        exit(1);
    }

    int backlog = 50;
    listen(sockfd, backlog);

    printf("Waiting for connection on port %d\n", port);

    /* Create a server socket for receiving connections */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("Can't create socket for receiving connections");
        exit(1);
    }

    /* Bind the server socket to the port */
    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
        perror("Can't bind to \n");
        exit(1);
    }

    /* Listen for incoming connections */
    if (listen(sockfd, backlog) == -1) {
        perror("Can't listen on socket");
        exit(1);
    }

    /* Accept incoming connections */
    if (accept(sockfd, (struct sockaddr *)&client_addr, &clientLen) == -1) {
        perror("Can't accept connection");
        exit(1);
    }

    /* Set up the send buffer */
    sendbuf = malloc(packetSize);
    if (sendbuf == NULL) {
        perror("Error allocating sendbuffer");
        exit(1);
    }
if (listen(sockFd, 5) == -1) {
    perror("Listen");
    exit(1);
}

printf("\r\nTCPServer Waiting for client on port %d", port);
fflush(stdout);

while(acceptflag)
{
    datasockFd = accept(sockFd, (struct sockaddr *)&client_addr,&clientLen);
    if(datasockFd < 0 )
    {
        printf("\r\n Accept Error \n\n");
        exit(1);
    }
    printf("\r\nReceived connection from :%s , %d \n", inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));

    while(recvflag)
    {
        addr_len = sizeof their_addr;
        if (((numbytes = recv(datasockFd, buf, TX_PACKET_MAX_SIZE , 0)) == -1)
        {
            perror("TCP recv error ");
            exit(1);
        }
        total_no_of_packets++;
        if (0==length)
        {
            GetSystemTimeAsFileTime(&sysTime);
            startTime = sysTime.dwHighDateTime;
            startTime = (startTime<<32) | sysTime.dwLowDateTime; /* number of 100-nanosecond intervals */
        }
        length+=numbytes;
        buf[numbytes] = '\0';

        FD_ZERO(&rfds);
        FD_SET(datasockFd, &rfds);
        retval = select(datasockFd+1, &rfds, NULL, NULL, &tv);
        if (((retval == 0) || (numbytes == 0))
        {
            GetSystemTimeAsFileTime(&sysTime);
endTime = sysTime.dwHighDateTime;
endTime = (endTime<<32) | sysTime.dwLowDateTime; /*
number of 100-nanosecond intervals */
timeDurationInMicSec = (endTime - startTime) / (long)

endTime = (endTime<<32) | sysTime.dwLowDateTime;

/* Subtract the last time-out value */
if (timeDurationInMicSec)
{
    if (numbytes != 0)
        timeDurationInMicSec -= (long)(TIMEOUT_PERIOD_IN_SEC * 1000*1000));
}

printf("\r\n\n******************$Receive Statistics
Start *******************\r\n\n\n$Time taken                     :%ld u sec\r\n\n%"),timeDurationInMicSec);

/* Convert to bits */
totalTxBits = length*8;

if (timeDurationInMicSec)
{
    dataRate = (totalTxBits*1000*1000)/timeDurationInMicSec;
}

if (dataRate)
{
    printf("$Number of packets recieved :%lld\n",(total_no_of_packets));
    printf("$Total number of bytes recieved :
\r\n\n%",(length));

    printf("\r\n\n\r\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n%",dataRate);
    printf("\r\n\n\n\n\r\n\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n%",dataRate);
    printf("\r\n\n\n\n\r\n\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n%",dataRate/1024));
    printf("\r\n\n\n\n\r\n\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n%",dataRate/1024));

    if (0 != expected_length)
    {
        if (expected_length >= length)
        {
            printf("\r\n\n\n\n\r\n\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n%",dataRate);
            printf("$Data loss                       :%lld bytes ( %f percentage)",(expected_length - length),
\n\n\n\n\r\n\n\n\n\r\n\n\n\n\r\n\n\r\n\n\n\r\n\n\r\n\n\n%",((float)((expected_length - length)*100))/(float)expected_length);
        }
    }
    else
    {
        printf("$Error: Unable to calculate throughput.");
    }

    length = 0;
total_no_of_packets = 0;
GS2K Throughput Measurement Application Note

```c
printf("\r\n*****************************************************$Receive Statistics End
          *****************************************************\r\n")
break;
}
}
break;

printf("\r\nUsage: \r\n\t throughput.exe [Protocol] [IP Address] [port num] [<Number of Packets>] [<Packet Size(bytes)>]\r\n")
printf("\r\nExample: Protocol: 0 - UDP Client. 1 - UDP Server. 2 - TCP Client. 3 - TCP Server\r\n")
printf("1. UDP Server: throughput.exe 1 : Start UDP Server on default port 8000\r\n")
printf("2. UDP Server: throughput.exe 1 9999 : Start UDP Server on custom port 9999\r\n")
printf("2. UDP Client: throughput.exe 0 192.168.40.1 8000 100 1400\r\n")
printf("2. TCP Server: throughput.exe 3 : Start TCP Server on default port 8000\r\n")
printf("2. TCP Server: throughput.exe 3 8888 : Start TCP Server on custom port 8888\r\n")
printf("2. TCP Client: throughput.exe 2 192.168.40.1 8000 100 1400\r\n")
exit(1);
}
switch(atoi(argv[1]))
{
    // UDP Client
    case 0:
#define DEBUG_ENABLE
    printf("\r\n UDP Client: \r\n")
#undef DEBUG_ENABLE
    switch(argc)
    {
        case 6:
ipaddress = argv[2];
        port = atoi(argv[3]);
        numofpacket=atoi(argv[4]);
packetsize=atoi(argv[5]);
udp_client(ipaddress,port,numofpacket,packetsize,totalbytes);
```
break;

default:
    printf("\r\n Enter Parameters correctly ");
    exit(1);
    break;
}
break;

// UDP Server
    case 1:
        #ifdef DEBUG
            printf("\r\n UDP Server: \r\n");
        #endif
        switch(argc)
        {
            // UDP Server Default Port
            case 2:
                port = DEFAULT_PORT;
                strcpy(portNum_udpserver, MYPORT);

            udp_server(ipaddress, port, numofpacket, packetsize, totalbytes);
                break;

            case 3:
                port = atoi(argv[2]);
                strncpy(portNum_udpserver, argv[2], 8);

            udp_server(ipaddress, port, numofpacket, packetsize, totalbytes);
                break;

        default:
            printf("\r\n Enter Parameters correctly ");
            exit(1);
            break;

        }
    break;

    // TCP Client
    case 2:
        #ifdef DEBUG
            printf("\r\n TCP Client: \r\n");
        #endif
        switch(argc)
        {
            case 6:
                ipaddress = argv[2];
                port = atoi(argv[3]);
                numofpacket = atoi(argv[4]);
                packetsize = atoi(argv[5]);

            tcp_client(ipaddress, port, numofpacket, packetsize, totalbytes);
                break;

        default:
            printf("\r\n Enter Parameters correctly ");
            exit(1);

        }
break;
}
break;

// TCP Server

#define DEBUG_ENABLE

switch(argc)
{
    //TCP Server Default Port
    case 2:
        port = DEFAULT_PORT;
tcp_server(ipaddress,port,numofpacket,packetsize,totalbytes);
        break;
    case 3:
        port = atoi(argv[2]);
tcp_server(ipaddress,port,numofpacket,packetsize,totalbytes);
        break;
    default:
        printf("\r\n Enter Parameters correctly ");
        exit(1);
        break;
}
break;

default:
    printf("\r\n Enter Parameters correctly ");
    exit(1);
    break;

}
return 1;
# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <errno.h>
# include <string.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <time.h>
# include <stdio.h>
# include <time.h>
# include <math.h>

#define CLOCK CLOCK_MONOTONIC
#define TX_PACKET_MAX_SIZE 1496

unsigned int fill_tx_data(char *data, unsigned int max_size)
{
    unsigned int i = 0;
    for (i = 0; i < max_size; i++)
    {
        *(data+i) = (unsigned char)i;
    }
}

int delayus(int Dlay) {
    double elapsed;
    struct timespec requestStart, requestEnd;
    #if 0
    int j;
    clock_gettime(CLOCK, &requestStart);
    temp = 0;
    for (j = 0; j < 4500; j++)
        temp += sin(j);
    clock_gettime(CLOCK, &requestEnd);
    elapsed = ((requestEnd.tv_sec - requestStart.tv_sec) / 1e-6
                + (requestEnd.tv_nsec - requestStart.tv_nsec) / 1e3);
    printf("Elapsed: %lf us
", elapsed);
    #endif
    clock_gettime(CLOCK, &requestStart);
    do
    {
        clock_gettime(CLOCK, &requestEnd);
    } while((elapsed = (requestEnd.tv_sec - requestStart.tv_sec) / 1e-6
             + (requestEnd.tv_nsec - requestStart.tv_nsec) / 1e3) <= Dlay);
}
int main(int argc, char**argv) {
    int sockfd, delay;
    struct sockaddr_in servaddr;
    char sendline[TX_PACKET_MAX_SIZE+1];
    long long packet_size;
    long long total_packets;
    double bandW;
    double tx_time_sec,temp_time;
    struct timespec tStart,tEnd;
    printf("\r\nargc=%d\r\n",argc);
    if (argc < 3) {
        printf("\r\nUsage:udpclient.exe <IP address> <port number> <no of packets> <packet size> <bandwidth> <TX time>\n")
        exit(1);
    }
    if(argc == 5) {
        total_packets=atoi(argv[3]);
        packet_size=atoi(argv[4]);
        bandW = 25;
    }
    if(argc == 6) {
        total_packets=atoi(argv[3]);
        packet_size=atoi(argv[4]);
        bandW = atof(argv[5]);
    }
    if(argc == 7) {
        total_packets=atoi(argv[3]);
        packet_size=atoi(argv[4]);
        bandW = atof(argv[5]);
        tx_time_sec = atof(argv[6]);
    } else {
        // infinite send
        total_packets = 0xFFFFFFFFFFFF;
        packet_size = TX_PACKET_MAX_SIZE;
    }
    // delay is added to generate packets as per required bandwidth.
    if (bandW == 25) delay = 400;
    else if (bandW == 20) delay = 500;
    else if (bandW == 18) delay = 600;
    else if (bandW == 15) delay = 750;
    else if (bandW == 12) delay = 900;
    else if (bandW == 10) delay = 1000;
    else if (bandW == 7) delay = 1600;
    else if (bandW == 5) delay = 2000;
    else if (bandW == 3) delay = 3500;
    else if (bandW == 1) delay = 11000;
    else if (bandW == 0.5) delay = 22000;
    else if (bandW == 0.3) delay = 37000;
    else if (bandW == 0.1) delay = 110000;
    else delay = 500;
}
sockfd = socket(AF_INET, SOCK_DGRAM, 0);

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(atoi(argv[2]));

fill_tx_data(sendline, packet_size);

printf("Port number:%d \n",atoi(argv[2]));
printf("Total tx packets:%lld size :%lld \n",total_packets,pACKET_SIZE);
clock_gettime(CLOCK, &tStart);
temp_time = 0;
while (temp_time < tx_time_sec)
{
    sendto(sockfd,sendline,packet_size,0,
        (struct sockaddr *)&servaddr,sizeof(servaddr));

delayus(delay);
clock_gettime(CLOCK, &tEnd);

temp_time = tEnd.tv_sec - tStart.tv_sec;
}
printf("End of Transmission\n");
}