Project Report

State Route 99 Traffic Signs:

Software Interface

Performed by

Sound Transit
Research and Technology

and

Stuart Mclean, Daniel J. Dailey and Fredrick W. Cathey
University of Washington
ITS Research Program
Department of Electrical Engineering
Box 352500
Seattle, WA 98195-2500

and

Washington State Transportation Center (TRAC)

August 2004

Table of Contents

List of Figures

1. Introduction

This report consists of four sections. The first section provides an overview of the software constructed to service the signs, a description of how the software works, and how to configure the application. The second section details an experiment requested by Sound Transit in October of 2003. The third section describes results from an experiment in January 2004. The fourth section gives the results of an experiment in April and May of 2004.

2. Software Design

This section details the software components that support the delivery and display of departure predictions about Metro King County transit vehicles to LED signs deployed at the roadside bus shelters.

2.1 Project Overview

The ‘sign server’ suite is a set of Java programs used to manage/control/administer communications between a central controlling server application and many Dynamic Message Sign devices sourced from Hitech Electronic Displays (hitechled.com). The software was developed by the Intelligent Transportation Systems research program at the University of Washington using the Java programming language. It runs on both Windows and Unix platforms. The primary application is the server application itself. As of May 2004, the server application is running on a Linux PC within the ITS/UW network.

Each Hitech sign consists of a 72 by 8 array of LED pixels for text display, plus a built-in wireless modem for communication over the Internet. The CDMA modem is supplied by Airlink Communications (airlink.com). Each sign is packaged in a weatherproof metal box for outside installation and unattended operation. Due to the pixel configuration and supported fonts, only one line of text of about 12 characters can be displayed on each sign at one time. The signs cycle through a series of messages in order to provide a variety of information. It should be noted that the Hitech signs in use for this project are not in any way tailored to the transit data application described here. Rather, they accept simple text strings like “Hello World.” Only the server application knows that the textual data sent to the sign, e.g., “Next Arrival in 3 Mins,” relates to transit vehicle movements.

As of May 2004, six signs have been installed at bus stops operated by the Metro King County transit agency in the Seattle, Washington metropolitan area. The signs have been installed one per location at sites chosen by MetroKC. The signs display real-time arrival and departure information for buses scheduled to service each stop.

Communication between each sign and the single server application is via TCP/IP using the signs' built-in modem. The server maintains a TCP socket connection to each sign.

Each Hitech sign has programmed into its non-volatile memory a string-valued 'serial number,' which it announces to the server upon connection establishment. This is the ONLY reliable remote means of identifying a sign. The IP address assigned to the modem cannot be relied upon, since static IP is not guaranteed on the CDMA network used by the sign/modem units. Accurate logging of sign serial numbers is thus important as the signs are installed in the field. The list of serial numbers is a central part of the overall server configuration.

ITS/UW has already developed software for calculation and public dissemination of real-time bus movement information for the King County region. The web site http://mybus.org provides estimated departure times for all Metro King County buses at all scheduled event locations. These locations do not directly correspond to bus stop locations but are good approximations. The sign server application described here taps into this data feed in order to provide each of the signs with bus movement information relevant for that sign location. For example, for the sign installed at the northbound bus shelter at the intersection of Aurora Ave N and N 85th Street in Seattle, the server is configured to send data to that sign for just those buses which serve that stop.

2.2 Data Flow + System Architecture

For the Metro King County transit agency, ITS/UW receives real-time automatic vehicle location (AVL) data reports directly from MetroKC's AVL system. Reports identify the vehicle and block of scheduled work the vehicle is performing and how far into the block the vehicle has reached. ITS/UW has defined a 'standard' AVL report, containing many more fields than are currently provided by MetroKC's AVL system. Thus, ITS/UW processes the incoming AVL data stream and expands the data reports into new ones containing all fields required by this 'standard' report.

Once in this standard format, AVL reports are further processed by a set of ITS/UW applications known collectively as 'Mybus.' First, a 'predictor' application estimates future bus movements, and, in particular, arrival/departure times at specific points in each block of work. Output of this predictor is maintained in a second Mybus application called the ‘Store.’ The Store is effectively an in-memory database system. It makes available the latest prediction about each scheduled event and can be queried by both event location and block id keys.

The Mybus Store has many clients. The Mybus web site application (http://mybus.org) is the primary consumer of the store data. For each request by a Mybus web site client, normally via a web browser, the web site queries the Store for the latest predicted times for bus arrivals/departures. Parameters in the query include the geographic location of interest.

The signserver application is another client of the Store. One important difference between the web site and the sign server system is that the web site is client driven while the sign server is server driven. For the web site, data is pulled from the system via a browser initiated HTTP request. In the sign server application, the Hitech signs are passive. The server schedules internally the work to be done to update each sign at the appropriate time. When a sign needs updating, the server queries the Mybus store for new data and formats the data for transmission to the sign using the proprietary encoding supported by Hitech’s signs. The server then pushes the data to the sign, which should acknowledge receipt of the data.

2.3 Software Components

In the notes below we assume that a copy of the sign server software is available locally on a Unix host. For a Windows installation, replace any shell script instructions with those appropriate for batch files.

We shall refer to the root of the software installation as HITECH_HOME. All applications under this root are written in 100% pure Java and thus will run on any platform with a suitable Java Virtual Machine (JVM). In this directory, the following sub-directories should exist:

- *server* – This directory holds the server application code and launch files. Also in this directory tree are logs of data sent between server and signs. A ‘config’ sub-directory contains the tools needed to re-configure the server application should the transit schedule change, which, for the MetroKC agency, occurs approximately three times per year.

- *conf* – This is the main directory that holds all text files which describe exactly for each sign what sequence of text strings should be sent to the sign and at what intervals in time. The XML format is used to describe the message sequences.
- *hup* – The hup program is used to communicate to the server that one or more text files in ../conf have been changed and that the server should re-process them accordingly. Use of a hup mechanism avoids having to shutdown and restart the server when any part of the configuration changes.
- *emulator* – This is a graphical program which emulates a real Hitech sign. Properties stored in a local configuration file mimic the non-volatile memory state variables of a real sign. A further sub-directory named ‘groups’ provides a means to run N emulators simultaneously.
- *GUI* – This is a graphical user console from which simple text strings can be sent to a chosen set of signs. It is envisaged as a tool for managing a sign during exceptional conditions where the regular bus schedule does not apply, e.g., ‘Snow Routes In Operation.’ It may be further developed into a more fully featured control application in the future.

2.4 The Server Application

The server directory contains the following files:

1. *server.[sh|bat]* – launch shell script/batch file
2. *server.prp* – example properties file with main server parameters
3. *stores.prp* – example properties file describing Mybus store access parameters
4. *its.app.mybus.vms.hitech.server.jar* – application code

where files ending in *.prp* are text readable properties files and *.jar* files are a compressed archive file containing Java class files.

The server launch files refer to both the jar file above and a second jar file in order to complete the classpath needed to run the application. The second jar contains a number of resources describing aspects of the transit schedule needed by the server application. For a transit schedule named ‘schedule’, this second jar would be located under $HITECH_HOME/server/config/schedule.jar. This file would then be referenced in the server launch file. See Section 2.11 on schedule shakeup below.

2.4.1 Server Startup Parameters

The server application accepts zero or one command line argument(s). If one is given, it is expected to name a local properties file. If no argument is given, a default properties resource is located in the server jar file itself. Configurable properties found in the local properties file are:

- *confhome* – This is the directory root for configuration files. If not supplied, defaults to ../conf, which is the correct value in all but server testing/debugging cases.
- *requestport* – This is the tcp socket on which to listen for connections from Hitech sign/modem devices. As of May 2004, all devices connect to port 3001 (on host 128.95.29.20, signserver.its.washington.edu). There is no default value. If a properties file is supplied, this property must be defined.
- *responsetimeoutsecs* – This is the number of seconds the server will block on a socket read when trying to read data from the sign/modem. Defaults to 0, which means block forever, if not supplied.
- *stores* – This is the file name containing Mybus store access parameters. Defaults to stores.prp if not supplied.
- *hups* – This is a whitespace-separated list of IP names and/or addresses from which the server will accept hup messages and re-process configuration files. Names/ addresses can be complete, e.g., 128.95.29.1, troika.its.washington.edu,

or wild-card matched by the * token, e.g., 128.95.29.*, 140.142.*, *.its.washington.edu, washington.edu. It defaults to the single entry ‘localhost’ if not supplied. This is the most secure option. Remote hupping has security implications since no authorization/authentication mechanism is currently built into the server.

The default properties resource, used if no prp file is supplied on the command line, defines the following properties:

- confhome : ../conf
- requestport : 3001
- responsetimeout : 0

To run the server:

- cd to $HITECH_HOME/server
- server.[bat|sh] [prpFile]

The server will start by reading appropriate configuration files (see Section 2.5) and then listens on the given TCP socket for remote sign devices to establish connections. In order to test the server, a sign emulator application is included in the distribution (see Section 2.7).

2.4.2 Server Logging

Various levels of logging have been created within the server application itself to document connection behavior due to problems experienced with the sign/modem connections to the server. Three different logs are maintained by the server application for each sign that communicates with the server (the server uses the signs serial numbers for logging purposes, so distinct signs (and/or emulators, see Section 2.7 below) should have distinct serial numbers).

The log files are found under $HITECH_HOME/server/logs. The following directories are maintained:

1. *connections* – This directory contains text files that document connection establishments and connection failures for each individual serial number. All entries are timestamped. Statistics regarding connection uptime, down time,

number of connections during any period, etc., can thus be derived from these logs.

Entries into this log are written by the server when the following occurs:

- *validation* – The server accepts a TCP connection from the sign/modem and receives a successful serial number response from the sign upon request. At this point, a connection to an identified device has been established.

- *error* - While reading or writing data bytes to/from the TCP socket, the server encountered an I/O error on the socket. As of May 2004, no write error has ever been seen. All errors are due to read failures. The underlying operating system which the server runs on receives TCP packets (segments). These packets request either a connection reset (TCP RST flag set) or a closure of the socket by the remote end (TCP FIN flag set). Either scenario results in the TCP connection being lost. It is not clear why these segments arrive at the server machine or from which entity in the network (ISP, modem) they are sent.

2. *socket* – This directory contains time stamped logs of every byte written and read by the server with respect to a single sign. For reads, a timestamp is logged before the server issues the blocking read call. Once a single byte is read, that is logged too. In addition to logging every byte of all the text strings sent by the server to the sign, this log contains all the bytes used in the message encoding used by Hitech, the sign vendor.

3. *goaltimes* – This directory contains a higher level log for debugging the Mybus-derived bus data strings sent to the sign. For example, this log shows HOW the string ‘2 & 15 Mins’ was calculated, given the state of the Mybus store at any moment in time. Given the data available in the Store, a relatively complex algorithm is applied to derive the final message, and this log is a view onto this algorithm. Factors involved include early vs. late running buses, the absence of good AVL data for a vehicle, etc.

2.5 Server Configuration

The sequence of messages sent from the server to any sign/sign emulator that connects to that server can be partitioned into two distinct sets/phases. These are the initial diagnostics phase, followed by the logically non-terminating schedule phase. Only when the TCP connection is lost does the schedule phase end. At this point, the server returns to listening for a new connection. When established, the whole ‘diagnostics followed by schedule phase’ cycle starts over.

2.5.1 Diagnostics Phase

The first phase is the ‘diagnostics’ phase. When a sign connects and is identified via its serial number, a sequence of ‘diagnostics’ messages are sent to the sign. These meta-data type message affect how the sign operates. For the Hitech sign, such diagnostics include:

- Setting the text font size.
- Setting the pixel brightness (intensity). This is either constant throughout the day or at various levels for various specified periods. Typically, the display is to be brightest during daylight hours and dimmer during the night.
- Setting timeout values which drive re-connection logic in the sign, e.g., if the sign has not received data in N seconds, assume there is a problem with either the network and/or the server and re-connect.
- Setting a default text message for the sign to display during the periods when a connection is not in place.
- Setting a ‘modem messages’ debug flag so that the sign will display the series of AT commands it issues to the modem to initiate a connection with the server. In the field, this would be set to false.

The sequence of diagnostic messages sent to each sign is independent of the sequence to any other sign, although typically a single diagnostics sequence is applicable to all signs. A sequence of diagnostics messages can be of length zero.

2.5.2 Schedule Phase

The second, and most important phase, of the server to sign message communications is the so-called schedule phase. The schedule phase is where the ‘schedule’ of instructions describing the sequencing and content of the text messages to be sent to the sign for display is read and interpreted by the server. The term schedule here pertains to a ‘recipe,’ ‘plan,’ or ‘itinerary’ and should not to be confused with the notion of a transit schedule. A typical message schedule might be ‘Route 11 to Downtown,’ ‘Next Bus In,’ ‘4 and 10 Minutes,’ ‘Time Now Is 12:20,’ at which point the schedule loops back to the top element.

Assuming that signs are installed at distinct locations in the field, it is highly unlikely in normal operation that two signs would be allocated to the same message schedule. The text strings sent in the message schedule typically contain bus route numbers, trip destinations, etc. (properties derived form the transit schedule) and times of next arrival/departures for buses serving that location. Even on opposite sides of the roadway at a single intersection/cross-street, two signs would be served by schedules describing buses heading in opposite directions. However, under exceptional conditions, such as when all buses might be cancelled due to adverse weather conditions, ALL signs might be controlled by the SAME schedule of ‘Snowing,’ ‘All Buses Canceled,’ ‘Update Expected 6.00pm.’ Switching of schedules by an operator as conditions apply is detailed below.

We define the term ‘configuration’ to mean the complete description of all diagnostics plus all schedule messages to be sent to ALL signs which the server is expected to administer/control. A configuration is given a local name. For example, in a server system controlling 5 signs identified as S1, S2, S3, S4, S5, a typical configuration might be:

- name = ‘Default MetroKC Configuration’
- diagnostics sequence Da (consisting of D1,D2 where Di is a single diagnostic) to be applied (sent) to signs S1, S3 and S4. D1 = set brightness to max, D2 = set timeouts to 30 seconds.

- diagnostics sequence Db (D1,D3,D4) to be applied to signs S2, S5. D1 = set brightness to max, D3 = set default message to ‘Not Connected,’ D4 = set modem messages to ON.
- message schedule SC1 to be applied to sign S1. SC1 consists of 4 rotating messages: Route 358, Next Arrival, X and Y mins (X and Y filled in by Mybus logic!), Time Now T.
- message schedule SC2 to be applied to sign S2. SC2 consists of 6 rotating messages: Route 48, Next Arrival, X1 and Y1 Mins, Route 355, Next Arrival, X2 and Y2 mins
- schedule SC3 applied to sign S3
- schedule SC4 applied to sign S4
- schedule SC5 applied to sign S5

Then a configuration is fully specified by

1. name

2. set of sign to diagnostics sequence assignments, e.g.

```
S1 : Da
S2 : Db
S3 : Da
S4 : Da
S5 : Db
```

3. description of each diagnostics sequence, e.g:

```
Da = D1,D2
Db = D1,D3,D4
```

4. set of sign to schedule assignments, e.g.

```
S1 : SC1
S2 : SC2
S3 : SC3
S4 : SC4
S5 : SC5
```

5. description of each schedule, e.g.

```
SC1 = (‘Route 11 to Downtown’, ‘Next Bus In’, ‘4 and 10 Minutes’,
‘Time
```

```
        Now Is 12:20')
```

2.6 Configuration File System

A configuration is stored in the signserver system as a set of text files adhering to a rigid naming convention and file system structure. Under $HITECH_HOME is a directory called 'conf.' Under this directory all configurations a server might use are stored, one per directory. If different than ./conf, the server's 'confhome' property must be set accordingly. Thus, a configuration called 'MetroKC' would be stored in $HITECH_HOME/conf/MetroKC. A configuration is effectively named by its directory under conf. Other than this directory name, a configuration's name is used only for descriptive purposes.

For a configuration named CNAME, two files and two subdirectories must exist under $HITECH_HOME/conf/CNAME/. The files contain information relating to the assignment of sign serial numbers to diagnostics sequences (point 2 above in Section 2.5.2) and to the assignment of sign serial numbers to schedules (point 4 above). The subdirectories contain files which describe the actual diagnostics and schedule contents (points 3 and 5 above).

2.6.1 diagnostics.prp

All sign to diagnostics assignments are specified in $HITECH_HOME/conf/CNAME/diagnostics.prp. This is a text file of key, value entries satisfying the syntax of a Java properties object, i.e., a non-commented and non-empty line is of the form 'key : value.'

The keys in diagnostics.prp are the sign serial numbers. An entry is required for all sign devices under control of the server instance. For any given key, say serial number S1, the value string denotes an XML filename relative to $HITECH_HOME/conf (or other confhome if not 'conf'). Thus, a diagnostics.prp entry

S1 : metrokc/diagnostics/common.xml

tells the server that when sign S1 connects, the diagnostics sequence for that sign is described in file $HITECH_HOME/conf/metrokc/diagnostics/common.xml. The configuration containing this file is ‘metrokc,’ which in most cases is the current configuration being described (i.e., CNAME = ‘metrokc’).

2.6.2 schedules.prp

All sign to schedule assignments are specified in $HITECH_HOME/conf/ CNAME/schedules.prp. This is a text file of key, value entries satisfying the syntax of a Java properties object, i.e., a non-commented and non-empty line is of the form ‘key : value.’

The keys in schedules.prp are the sign serial numbers. An entry is required for all sign devices under control of the server instance. For any given key, say serial number S1, the value string denotes an XML filename relative to $HITECH_HOME/conf (or other confhome if not ‘conf’). Thus, a schedules.prp entry

S1 : metrokc/schedules/aurora85_north.xml

tells the server that when sign S1 connects, the schedule of messages to be applied to that sign is described in file $HITECH_HOME/conf/metrokc/schedules/aurora85_north.xml. The configuration containing this file is ‘metrokc,’ which in most cases is the current configuration being described (i.e. CNAME = ‘metrokc’). The file name, e.g., aurora85_north.xml, is arbitrary, but would be named to reflect the physical location at which the sign is installed. Its contents would then describe messages relating to buses passing by that location (see Section 2.6.3 below).

2.6.3 Diagnostics Files

The sequence of diagnostics messages sent to a sign is described using xml text files. The xml schema for valid diagnostics files is included in the software distribution. The diagnostics which can be applied to a Hitech sign are detailed in the Hitech protocol document, a copy of which is also included in the distribution.

A typical diagnostics file is:

```
<configuration id=  “common  “>

<brightness-schedule default=  “8  “>
```

```
<period starthour=  “7  “ startmin=  “0  “ stophour=  “16  “ stopmin=
“0  “ level=  “10  “/>
</brightness-schedule>

<font value=  “6x8  “/>

<offline-message scrolling=  “false  “ text=  “RESETTING  “/>

<timeout-values hitech=  “3  “ modem=  “30  “ online=  “30  “ offline=
“30  “/>

<modem-messages value=  “false  “/>

<set-time/>

</configuration>
```

This diagnostics sequence contains 5 diagnostics. They are transmitted to the sign in the order given.

1. *brightness-schedule*: Up to 8 periods can be specified and would be as follows: Between 7am and 4pm (16:00) set the pixel brightness at level 10, the maximum. At all other times, the brightness should be the default of 8.
2. *font-value*: This is not a diagnostic message per se, but it is packaged as control information with each text message. Thus, the font value is recorded by the server for each sign and that value is then sent with each text message in the schedule phase. (This appears to be the only way to set the sign’s font.) Note that currently this option has no effect, e.g., all signs’ fonts are currently fixed at 6x8 monospaced. In testing, it was agreed that this is the only legible font for the current signs.
3. *offline-message*: This is the text the sign should display when the connection to the sign server is lost.
4. *timeout-values*: For Hitech and modem values, this information is in the protocol document included with the distribution. The online value specifies how long the sign should wait to receive new data from the server when the TCP/IP connection is in place. Given that the message schedules used send new text every 3 or 5 seconds, 30 seconds is plenty. If the online timeout

expires, the sign will tell the modem to re-dial. The offline timeout specifies how long to wait between reconnection attempts.

5. *modem-messages*: This is a debugingoption only. If on (true), the sign will display the AT commands it uses to control the modem. For an installed sign, it would be off (false).

6. *set-time*: This sets the internal clock in the sign. The sign uses its own clock for various functions, including switching brightness periods. Setting the sign clock upon connection establishment should reduce any clock drift (assuming the server's clock is correct).

One other diagnostic of interest is:

7. *set-dial-number*: This is used for debugging/installation only. It sets the IP address and port which the sign should dial up when it is next disconnected. It enables the server to 'offload' the sign to a new server. Care should be used with this command. If there is no control over the new IP address, there may be no remote means for retrieving control of the sign/modem.

2.6.4 Schedule Files

The cycling sequence of text messages sent to a sign is described using xml text files. The xml schema for valid schedule files is included in the software distribution as $HITECH_HOME/doc/schedule.xsd. This schema is more application-oriented than the diagnostics schema, which was entirely Hitech sign specific. The schedule schema describes exactly what Mybus data should be calculated, how it should be formatted, and when it should be sent.

A sample schedule file is:

```
<schedule id=  “aurora46-north  “>
 <description>Aurora OB @ 46th</description>
<tasks run=  “forever  “>

<events duration=  “5  “>
<text value=  “358 AUR VILL  “ duration=  “5  “/>
<text value=  “NEXT ARRIVAL  “ duration=  “3  “/>

<retrieve storedescriptor=  “metrokc  “ location=  “1143  “
past=  “30  “ future=  “30  “ errortext=  “No Server  “/>
```

```
<select>
<route value=  “358  “/>
</select>

<display trimChars=  “&amp;   “>
<countdown units=  “mins  “ text=  “%1 &amp; %2  “ trailing=  “ MIN  “>
<imminent text=  “DUE  “ collapseMultiples=  “true  “/>
</countdown>
</display>

</events>


<!-- show the current time for 5 seconds -->
<time duration=  “5  “/>

</tasks>

</schedule>
```

The schedule ‘id’ attribute is not read by the server, but is used by the GUI app for descriptive purposes (see Section 2.9). The main element is the ‘tasks’ element, whose attribute ‘run’ should always be set to forever so that the enclosed task list repeats without termination. Allowable task types, which may appear in any order and any number of times, are:

1. *text*: This displays the text value for the specified duration, in seconds. The text will be sent to the sign. The server will then wait ‘duration’ seconds before sending the next string.

    Note that there are NO text tasks at the top level in the above schedule snippet. The <text> elements shown are within an <event> element (see number 3 below more details of the event element). The difference between the two is that a top level text task (i.e., one whose parent element is the <tasks> element) results in an unconditional transmission of the text to the sign. A text task within an <events> element is sent only if the string resulting from the event calculation is non-empty, i.e., there are some buses due. There is no point showing ‘Next 358 to Downtown’ if there are no subsequent bus times to be displayed.

2. t*ime*: This displays the current time and wait ‘duration’ seconds. The default format for the time string is currently ‘h:mma,’ displayed as ‘12:34PM’.

3. *events*: This is the most important element in the schedule, since it describes the Mybus-derived bus data text which the sign is to display. The events element has four sub-element types:

    a) *text elements* – These are text strings to be sent to the sign only if the events task itself computes some bus time text, given the state of the Mybus store and selection criteria of events to show.

    b) *retrieve element* - The Storedescriptor attribute maps to a set of store access parameters in the server file stores.prp (see Section 2.4.1). The location attribute specifies a location id from the transit schedule (currently all locations must be timepoints in the schedule). This id is included in the Store request message. The Store will respond with all events at that location whose scheduled event time (NOT whose estimated actual time) falls in the window NOW-past, NOW+future, where past and future are additional required attributes of the retrieve element (units are minutes). In the above snippet, predictions for all events in the Store denoted by ‘metrokc’ (maps to stores.prp key) at location id 1143 whose scheduled time is NOW-30 mins to NOW+30 mins would be retrieved. Note that ALL event predictions are retrieved for all scheduled buses on all routes, including deadhead events, end of trip, departed buses, etc. The Mybus store has no filtering logic built in. This must be done by the client; in this case, the selection element ‘next.’

    c) *select element* – This enables filters to be applied to the event prediction list returned from the Store. Provided filters are:

        - a route filter, used above, to select only routes 358 – xml element is <route>

        - a direction filter (values are “inbound” and “outbound”) – xml element is <direction>

- a chained trip destination filter – xml element is <destination>

    Any number of selection filters can be applied. They are AND-ed together, so a scheduled event (and its associated prediction) must satisfy all the filters to be considered for display on the sign.

    Implicit in the selection logic is the rule that departed buses will not be considered for display, since this is meaningless on a countdown style display. Problems do arise if there is no real-time information on a bus. Then the schedule time is used as the best estimate of the arrival (i.e., the schedule acts as a predictor). An according-to-schedule departed bus is processed as a departure too. The complete algorithm for event selection and estimated arrival time is available from ITS/UW.

d) *display element* – The attribute 'trimChars' denotes which characters in the display string should be trimmed from both left and right of the result string if they appear at either end. It is typically used to trim the conjunction (e,g, '&', as used in '3 & 5 Mins,') if only one bus is due. Without trimming, the result string would be '3 &', which makes no sense. Note that in xml the '&' character is special and must be escaped as '&amp;' to appear as data.

    The countdown element is currently the only allowed sub-element of the display element. It is used so that the sign display counts down the impending bus arrival, i.e., 4 Mins, 3 Mins, 2 Mins. Attributes are:

    - *units* – This is almost always mins. A seconds-based countdown is impractical.
    - *text* – This is a format string. Most characters in the text string appear in the final result string of the events task for transmission to the sign. The actual bus times are inserted

wherever a %N appears in the text string. Thus, %1 is replaced with the estimated time of the next bus (NOT the next scheduled time but the next ACTUAL time), %2 by the next but one, etc. Due to the Hitech signs having just 12 characters, no more than 2 bus arrivals can realistically be shown at once.

- *trailing* – This is constant text to append to the result of text above.
- *leading* (not included above) – This is constant text to be prepended to the text result.

The <imminent> sub-element of <display> is included by request from Metro KC. If a bus arrival is less than 2 minutes from the sign, the arrival is said to be ‘imminent.’ A special imminent text (“DUE” is used above) is substituted for the actual minute value. Thus ‘1’ and ‘2’ become ‘Due.’ Multiple imminent arrivals, which can occur if the text string is ‘%1 & %2’ and both the next two arrivals are imminent, would normally be shown as ‘Due & Due.’ Setting ‘collapseMultiples’ to true shortens the result to just ‘Due.’

Example display configurations when the next bus is imminent (less than 2 minutess away) and the following one is estimated to arrive in 10 minutes are:

```
text=  “%1 & %2  “ ->   “Due & 10  “
text=  “%1 & %2  “ trailing=  “Min  “ ->   “Due & 10 Min  “
leading=  “Second Bus:  “ text=  “%2  “ trailing=  “Min  “ ->   “Second
Bus: 10 Min”
```

2.6.5 The Active Configuration

In the $HITECH_HOME/conf directory (or some other directory if confhome is set otherwise), there exists a text file called ‘active’. Its sole content is the name of the configuration which the server should use on startup. Using a runtime notification mechanism (see hup described in Section 2.8 below), the operator can edit this file and cause the server to switch entire configurations. Under normal operation, the active file would be left as is. As of May 2004, the configuration describing the six Hitech signs installed and operating for MetroKC has been given the name ‘mybus.’ Thus, the active file contains just the word ‘mybus.’

2.6.5.1 The Signs File

Also in $HITECH_HOME/conf is a text file named ‘signs.’ This file serves as the central documentation for various aspects of each Hitech sign/modem unit. It is expected that maintenance/updates of this file will always be a manual process. Properties include:

- serial number
- install location
- ISP provider. As of May 2004, the six installed signs have Sprint and Verizon providing the network accounts, with three modems assigned to each provider.
- modem phone number, used as an extra means of identifying/debugging the modems
- esn, as per phone number. A hardware identification of a modem, like an Ethernet address.

The signs file is NOT used by the main server application, nor is it referenced anywhere in a configuration. The only software component in the distribution which reads the signs file is the GUI application, see Section 2.9. No application writes the signs file.

2.7 The Emulator Application

In order to test the server application and its various required runtime files, a sign emulator application is included in the software distributon. The emulator mimics as closely as possible a real Hitech sign (i.e., it uses the same proprietary protocol for message encoding, has the same timeout functionality, etc). Indeed, the server cannot distinguish a real Hitech sign from an emulator. In the $HITECH_HOME/emulator directory, the following files should exist:

- emulator.[sh|bat] - launch shell script/batch file.

- emulator.prp - example properties file with emulator parameters
- its.app.mybus.vms.hitech.emulator.jar - application code

2.7.2 Emulator Start Configuration

The sign emulator application accepts zero or one command line argument(s). If one is given, it is expected to name a local properties file. If no argument is given, a default properties resource is located in the emulator jar file itself. Configurable properties are:

- controllerhost : IP hostname or address of server application
- controllerport : tcp socket on which server is listening for connections
- serialnumber : the string-valued serial number of the emulator

To run the emulator:

- cd to $HITECH_HOME/emulator
- emulator.[bat|sh] [prpFile]

A graphics window should appear on the desktop. Once the emulator connects to the server application and identifies itself via its serial number, the server will send text strings based upon the server's configuration. The strings are displayed in the emulator window. A typical sequence of message strings might be

358 Downtown	- displayed for 5 seconds
Next Arrival	- displayed for 3 seconds
3 & 15 Mins	- displayed for 5 seconds
11:54AM	- displayed for 5 seconds.

It would then loop back to the first message.

2.8 The Hup Application

As described above, the main sign server application reads text configuration files at runtime in order to derive the text messages appropriate for each sign. Should the server operator/maintainer wish to change the server behavior, this is achieved via a two-step process:

1. Edit the appropriate file(s) in a regular text editor.
2. Send the server notification that those file(s) have changed and the contents re-processed.

The notification is done via the 'hup' application included in the software distribution. The hup is achieved by the hup application sending a UDP packet to the host/port on which the server is listening for hup messages. The hup mechanism used here is NOT a Unix-style signal mechanism since these are not available in a cross-platform Java environment.

In $HITECH_HOME/hup, the following files should exist:

- hup.[sh|bat] - launch shell script/batch file
- its.app.mybus.vms.hitech.hup.jar - application code

The hup program is driven by its command line arguments. The command line grammar is:

```
hup [-h host] [-p port]
(active|diagnostics-assignments|schedule-assignments|diagnostics-
contents
FILE|schedule-contents FILE)
```

In general, it is likely that the server would be hupped from the localhost, i.e., the server and hup apps will reside and be run on the same machine. Thus, the -h and -p options to the hup are not used in practice. If remote hupping is required, the -h option should be used to identify the server host.

The hup program accepts five forms of command line input:

1. *hup active* – This notifies the server that the active configuration has changed. This hup would be used after the 'active' file (under conf) has been edited, with a new active configuration name replacing the existing active name. Upon receipt of this hup, the server will read the 'active' file to identify the to-be activated

configuration and process all diagnostics and schedule files in that configuration. This is done without losing socket connections to any connected signs.

2. *hup diagnostics-assignments* – This notifies the server that the diagnostics.prp file has changed within the active configuration. The server re-reads the file and calculates which sign-to-diagnostics assignments have changed. Diagnostic sequences are applied to any connected signs for which a diagnostics assignment is seen to have changed.

3. *hup schedule-assignments* – This notifies the server that the schedules.prp file has changed within the active configuration. The server re-reads the file and calculates which sign-to-schedule assignments have changed. New message schedules are communicated to any connected signs for which a schedule assignment is seen to have changed.

4. *hup diagnostics-contents PATH_TO_DIAG_FILE* – This notifies the server that the diagnostics sequence description file HITECH_HOME/conf/PATH_TO_DIAG_FILE has changed and should be re-processed. Typically, the FILE_NAME would be CNAME/diagnostics/file.xml for an active configuration named CNAME. The path MUST be rooted at $HITECH_HOME/conf. It is NOT sufficient to provide D1.xml as the hup argument; it must be CNAME/diagnostics/D1.xml.

   Any connected sign currently assigned to the newly altered diagnostics sequence will be sent the new diagnostics sequence.

5. *hup schedule-contents PATH_TO_SCHED_FILE* – This notifies the server that the message schedule description file $HITECH_HOME/conf/PATH_TO_SCHED_FILE has changed and should be re-processed. Typically, the FILE_NAME would be CNAME/schedules/file.xml for an active configuration named CNAME. The path MUST be rooted at $HITECH_HOME/conf. It is NOT sufficient to provide SC1.xml as the hup argument, it must be CNAME/schedules/SC1.xml.

Any connected sign currently assigned to the newly altered message schedule (likely to be zero or one in number) will be sent the new schedule message sequence.

2.9 The GUI Application

Included in the software distribution under $HITECH_HOME/gui is a graphical operator console. The console allows an operator to override the normal behavior of the sign server. Currently, the GUI console supports the following functions:

- inspect various properties of each Hitech sign – This includes the serial number, installed location, ISP, modem netphone, esn, etc. These values are read directly from the 'signs' file in ../conf.
- switch between active configurations
- build new configurations – A limitation is that the auto-generated xml files for message schedules can contain just a single message.
- delete configurations, except for those identified as read-only
- identify groups of signs and save the group under a group name for later referencing.

The primary task which the GUI *cannot* yet do is to actually build xml schedule files. Due to the extensive schema and the myriad configurations of message schedules, it is arguable if a form-based input system would be feasible.

In the $HITECH_HOME/gui, the following files should exist:

- gui.[sh|bat] - launch shell script/batch file
- gui.prp - example properties file with GUI parameters
- its.app.mybus.vms.hitech.gui.jar - application code

2.10 GUI Start Configuration

The GUI application accepts zero or one command line argument(s). If one is given, it is expected to name a local properties file. If no argument is given, a default properties resource is located in the GUI jar file itself. Configurable properties are:

- *server* : IP hostname or address of server application. Defaults to ‘localhost’ if not supplied.
- *serverport* : tcp socket on which server is listening for connections. Defaults to 3001 if not supplied.
- *confhome* : home directory for configurations. Defaults to ../conf is not supplied.

The default properties resource, used if no prp file is supplied on the command line, defines the above three property names with their default values as stated.

2.11 Schedule Shakeup

The server application relies on some files containing information derived from the operating transit schedule. Should the schedule ever change (for the Metro King County agency this occurs approximately every four months but can occur with higher frequency), these files must be rebuilt and the server restarted. Included in the software distribution is an Ant build script that can rebuild these files.

Reliance on this local file data is necessary since not all properties of a scheduled event are available in the prediction data retrieved from the Mybus store. For example, in the message schedule data files, the operator can specify that only “inbound” buses on some route R should be displayed on a sign. That sign would presumably be installed at a bus shelter serving inbound buses (Inbound as used by Metro King County means towards downtown Seattle and outbound is away from downtown Seattle.). Currently, the directonality (inbound vs. outbound) of a bus trip passing through a scheduled location is not a property which can be obtained from the Store’s prediction data. So a local file of ‘direction data’ maps each trip identifier (a field which IS provided in a prediction report) to its inbound or outbound direction.

It is assumed that the new transit schedule data is stored in an accessible sql database system for which a Java JDBC driver is available. The ITS/UW-defined ‘standard transit schema’ is assumed to be the sql schema used to store the data. This schema is described in the documents directory of this distribution, i.e., $HITECH_HOME/doc/transit_ddl.sql.

To build the required runtime data files for the server application, the following sequence applies:

- cd to the $HITECH_HOME/server/config directory
- edit the db.prp file so that the latest db properties are in place, esp dbschema.
- run Ant on the build.xml, i.e just ‘ant’ suffices.

The build procedure builds a jar file with the same name as the dbschema. So if the database transit schema name is ‘april2004,’ the file will be $HITECH_HOME/server/config/april2004.jar. This file must be included in the server app’s classpath:

- cd to $HITECH_HOME/server
- shutdown the server app (via ps + kill on Unix)
- edit the launch script so that ./config/april2004.jar is included in the classpath.
- restart the server.

3. MyBus Software Interface Project - SR-99 Highway Signage: October 2003

3.1 Project Outline/Overview

The Intelligent Transportation Systems (ITS) research group at the University of Washington (UW) has developed software which delivers real-time bus information to variable message signs (VMSs). The software integrates the ‘Mybus’ real-time data developed by the UW ITS research group with a VMS provided by Hitech LED Signs, Inc. Metro King County Transit owns the sign(s) and guides the project. VMSs are to be installed at bus stops in the King County, Washington, region.

The signs are to be located at bus stops. Communication between the central ‘server’ application and the signs is via a wireless communication link. Each Hitech sign incorporates a modem, provided by Airlink Systems, which handles wireless data transfer from the server to the sign. The modem uses the Sprint PCS network as its Internet service provider. Currently, the server application is hosted at ITS/UW though this is not a requirement of the system architecture.

In theory, the server and a sign should be able to transfer data at any time. The communication protocol chosen for the project is TCP/IP, a connection-oriented protocol widely used on the Internet. Upon powering up, the sign contacts the server, using a TCP/IP ‘active open’ connection establishment routine. The server performs ‘passive opens’ and accepts the incoming connection requests from the sign(s). Once a connection is established, data can independently flow in both directions.

The server application attempts to send textual data (i.e., ‘Next Route 44 Bus’ or ‘Will Arrive 3 Mins’) to the sign at set intervals, nominally every 5 seconds. The text is enclosed in a ‘message’ coded according to the Hitech protocol. This protocol is a binary data format used by Hitech to send data to and read data from a Hitech VMS.

In theory, the TCP/IP network connection between the server and a sign could remain established continuously, as long as both the server application and sign are both powered up and operating as designed. Unfortunately, it has been discovered that the connection between the server and the sign is lost intermittently. Thus, messages sent by the server do not reach the sign, and responses from the sign to the server (if any are sent) are not received by the server.

The purpose of the experiment described in this report is to quantify the state of the connection over an extended period, nominally seven days.

The aim of the experiment is to measure the robustness of the TCP/IP connection established between a Hitech sign, with its enclosed Airlink Raven CDMA modem, and a server application, which is hosted at the University of Washington. The server attempted to send Hitech protocol-formatted messages to the sign at set intervals (approximately every 5 seconds) for the duration of the experiment. The Hitech sign sends messages back to the server application, verifying that it has received the server’s message and acted on it accordingly (i.e., the given text was been displayed).

3.2 Experiment Conditions

The experiment started Wednesday, October 8, 2003, at 12 noon.

The experiment ended Wednesday, October 15, 2003 at 12 noon.

The duration of the experiment was 7 days, which is equal to 168 hours or 10,080 minutes or 604,800 seconds

3.2.1 Server Configuration

The server software application was run on the host nova.its.washington.edu (128.95.29.3). The operating system was Windows NT 4.0 (build 1381) with Service Pack 6. The server application was a Java program, and Sun Java Virtual Machine (JVM) version 1.3.1 was used to run the compiled Java code. The code was written to send data once, but the TCP/IP implementation within the operating system is set to a retransmission count of 10. This means that the host server will try for at least 5 minutes to get data to the sign. This time interval is sufficiently large enough that the sign will timeout. See Section 2.4 for further details on timeouts.

3.2.2 Sign Configuration

The sign was located in an office at the University of Washington for ease of visual monitoring. The sign display was visible every day to ITS staff during the approximate hours of 9 AM to 5:30 PM, with the exception of Saturday, October 11 and Sunday, October 12.

The sign was configured to display all text in a 6x8 mono-spaced font and to use a static (i.e., non-scrolling) text display mode. The sign has 72 LEDs across its width and can therefore display 12 text characters (i.e., 72 / 6) at one time while using this particular font.

The brightness schedule feature of the sign was employed. The means that the text can be displayed with varying brightness for user-specified periods of each day. Between 7 AM and 4 PM every day, the sign was to display text using brightness value 10 (brightest available). Between 4 PM and 7 AM, the brightness level was to be 5 (a medium value). The brightness schedule feature requires that the sign keep time. The 'set time' function was used every time the sign connected to the server so that the sign could be initialized with NTP (network time protocol) time.

3.2.3 Data Sent From Server To Sign

The data sent from the server to the sign during the experiment consisted of text that was provided by Metro King County Transit. The data sent was relevant to the westbound bus stop at the intersection of N 85th St and Aurora Ave N in Seattle. Route 48 and Route 355 buses pass through this location. Route 48 buses are scheduled to pass westbound through this point between 06:20 AM and 12:06 AM. Route 355 buses are scheduled to pass this point between 3:39 PM and 6:30 PM. The sequence of text messages sent to the sign is shown in Table 1.

Note that text strings 1 through 3 are sent only when Route 48 is scheduled within the next 30 minutes (i.e., between 05:50 AM and 12:06AM). At other times, this message set is not sent to the sign. Similarly, messages 4 through 7, relating to Route 355, are sent only between 3:09 PM and 6:30 PM. This means that between 12:06 AM and 5:50 AM, only message 8, the current time, is sent to the sign, at a rate of once every 5 seconds. Sending different messages at different times of the day affects the number of bytes being exchanged. This time-of-day-dependency should be considered when interpreting the number of bytes exchanged between the server and the sign.

Table 1: Sequence of Text Messages

Message	Display Time	Notes
48 LOYAL HTS	5 secs	
NEXT ARRIVAL	3 secs	
X & Y MIN	5 secs	X and Y are replaced by Mybus prediction data values
355 SHORL CC	5 secs	
EXPRESS	5 secs	
NEXT ARRIVAL	3 secs	
X & Y MIN	5 secs	X and Y are replaced by Mybus prediction data values
HH:MM AM|PM	5 secs	the current time

In the Hitech protocol design, the server is always the master and the sign is always the slave. The server initiates message sequences, and in this experiment, it sent text string data to the sign as described above. At the application level, this consists of a single Hitech message. The sign then acknowledges receipt of the message, sending a 2-byte response back to the server. Once it has acted upon the text message, the sign then sends an 'OK' message to the server. The server acknowledges the OK message with a 2-byte response back to the sign. The server then waits some period of time (3 or 5 seconds as described above) and repeats the cycle. At the Hitech protocol level, the data transfer therefore looks like this:

1. server to sign - 12 bytes (header) + 6 bytes (style and positioning of text) + text length bytes 'L.' So, 'Hello World' transfers 12 + 6 + 11 = 29 bytes. In general, 18 + L.
2. sign to server - 2 byte response
3. sign to server - 12 byte OK message (12 byte header + 0 bytes of data)
4. server to sign - 2 byte response
5. server waits some time, then goes to step 1

3.2.4 Timeouts

The sign was configured to timeout and re-initialize the modem if data were not received from the server in 30 seconds. This is the default value built into the sign firmware. This means that the sign would wait 30 seconds for the arrival of message 1, as detailed above. In normal operation, the 5-second server intervals between message transmissions means that this 30-second timeout would never occur. However, in the presence of network (or other) failures, the sign may not always receive the server-transmitted messages. Thus, a sign-initiated timeout could occur. When the sign times out on a read (defined as the act of attempting to read bytes across the connection), it re-initializes the modem. Upon re-initialization, the modem closes its side of the connection with the server. A connection closure of this type causes the modem's TCP implementation to send the server's TCP implementation a special TCP segment. Upon receipt, the server notifies the server application that there are no more data to be read. This results in the server trying to read the sign's response (message 2 above) but getting notification that the connection will never contain any further bytes to read. The server application then aborts its regular cycle, described above, and closes its side of the connection. At a later time, the sign attempts to re-establish a new connection with the server, and the whole cycle begins anew.

The server was configured to not timeout when reading data from the sign. That is, when the server issues a read call to obtain messages 2 and 3 from the sign (see above), it will wait forever for the bytes to arrive. Only an input/output error would cause the server to abort reading expected data from the sign. The connection closure described above is an example of this error, and it causes the server to abort reading any further data on the connection.

3.3 Experiment Results

The server remained running for the duration of the experiment. In other words, whenever the Hitech sign was trying to connect to the server after a connection failure, the server was always ready to accept the connection attempt. Further, any data sent by the sign were always read by the server.

As far as we aware (no notice of scheduled/unscheduled power outages in the office where the sign was located), the sign remained powered up for the duration of the experiment. Given that the server and the sign were both operational, any data transfer failures can be attributed to either a network failure, a logical error on the server, a logical error on the sign, or a combination of these. Since the server succeeds in most of its message transmissions, we have no reason to think the server's logic is erroneous. We have insufficient information about the modem and the network configurations to derive any further explanation of the problems.

Visual monitoring of the sign throughout the experiment verified that the brightness schedule feature functioned as expected, with brightness changes occurring within 1 minute of 4 PM (the 7 AM switch was, however, never visually monitored).

As expected, the network connection between the server and the sign failed several times. We define a 'connection' as the period of time from the instant *S* when the server accepts the sign's connection request to the instant *E* that a message sent from the server is known not to have been received by the sign. Due to the 30-second timeout built into the sign, the time *E* is actually 30 seconds earlier than when the server gets an input/output error performing its next read.

The 'results' from the experiment were collected in three files:

- *Server.log* - This is the server's main log and is maintained by the server application. It logs TCP/IP connection establishments with timestamps, connection breaks with timestamps, total bytes written, and reads per connection. This log forms the source of the primary results detailed later.
- *CM04280002.log* - (serial number of sign is CM04280002) This logs every byte written by the server to the sign, along with the write time, the time before every blocking read is called, and the byte value and time received for every byte read. It is maintained by the server application.
- *tcpdump log* - This is a log of every TCP segment originating from or destined for the server application. It is a low-level tool used to monitor communication between the server application and the sign. It contains useful information for diagnosing possible TCP implementation anomalies in the modem and/or Windows NT OS or any network component in between.

3.3.1 Connecting IP Address

In the seven-day experimental period, we logged 74 separate connections. The final connection remained in place past the experiment ending time. Thus, there were 73 connection failures.

Each time the sign connected to the server, the server logged the IP address of the host sending the request for connection. The IP address was 68.26.112.169 and remained constant in all 74 connections. This address resolves via DNS (Domain Name Server) to 000-160-536-area1.spcsdns.net. The name hitech2.eairlink.com also resolves to this IP address. In an earlier discussion, it was noted that a changing IP address for the sign may cause problems. We are unable to verify that this might be a problem since only a single IP address was recorded during the experiment.

3.3.2 Log Summaries and Results

3.3.2.1 Connections

Over the entire testing period, there were 74 connections. The total bytes sent from the server to the sign was 3205432 (3.057 MB), and the total bytes read by the server (from the sign) was 1551080 (1.479 MB).

The minimum duration time for a connection was 3.15 minutes and the maximum duration was 286.58 minutes (~4.8 hours). The median connection time was 125.96 minutes (~2 hours), and the mean connection time was 135.66 minutes (~2.3 hours). The standard deviation was 62.45 minutes (~1 hour). The percentage of uptime for the duration of the testing period was 98%.

Table 2 below details the statistics for connection times over the course of each day. The headings in the table are defined as follows:

• Count:	number of connections
• Min:	duration of shortest connection (minutes)
• Max:	duration of longest connection (minutes)
• Median:	median duration (minutes)
• Mean:	average duration (minutes)

- STD: standard deviation of duration (minutes)

Table 2: Connection Times by Day

Date	Count	Min	Max	Median	Mean	STD
Wed, 08	5	65.57	254.80	137.33	144.72	69.92
Thurs, 09	11	13.47	234.48	154.43	135.35	63.30
Fri, 10	11	3.15	231.83	121.50	127.60	68.37
Sat, 11	11	3.17	227.85	127.45	127.72	67.80
Sun, 12	10	71.43	286.58	139.44	153.79	59.19
Mon, 13	9	92.45	198.53	124.08	139.43	42.90
Tues, 14	13	3.63	276.58	116.42	119.39	65.68
Wed, 15	4	78.40	245.83	174.51	168.31	84.41

3.3.2.2 Down Periods Between Connections

Over the entire testing period, there were 73 down periods between connections. The minimum duration for a down period was 1.47 minutes. The maximum duration was 33.47 minutes, which occurred from 4:16 AM until 4:50 AM, Thursday, October 9, 2003. This down period of 33.47 minutes is far greater than any other down period. The second highest duration for a down period was 3.58 minutes. The median duration for down periods was 1.57 minutes, and the mean was 2.15 minutes. The standard deviation was 3.74 minutes. If the highest down period of 33.47 minutes is deleted from the data, than the mean is 1.71 minutes and the standard deviation is 0.44 minutes.

Table 3 below details the statistics for downtimes between connections over the course of each day. The headings in the table are defined as follows:

• Count:	number of down periods (no connection)
• Min:	duration of shortest down time (minutes)
• Max:	duration of longest down time (minutes)
• Median:	median down time duration (minutes)
• Mean:	average down time duration (minutes)
• STD:	standard deviation of down time duration (minutes)

Table 3: Downtimes Between Connections by Day

Date	Count	Min	Max	Median	Mean	STD
Wed, 08	4	1.57	2.67	1.59	1.85	0.54
Thurs, 09	10	1.55	33.47	1.57	4.97	10.03
Fri, 10	10	1.57	3.08	1.58	1.87	0.60
Sat, 11	10	1.55	1.73	1.58	1.59	0.05
Sun, 12	9	1.47	1.62	1.57	1.55	0.05
Mon, 13	8	1.57	2.93	1.57	1.74	0.48
Tues, 14	12	1.48	1.67	1.57	1.57	0.05
Wed, 15	2	1.57	1.57	1.57	1.57	0.00

Figure 1 shows connection behavior over the duration of the testing period, 7 days from noon Wednesday, October 8, 2003 to noon Wednesday, October 15, 2003. It shows length of each connection and downtime in hours. Figures 2 shows the connection data broken down by day. Figure 3 shows only the connection times in hours, and Figure 4 shows only the downtimes in minutes. This shows that the 33 minute downtime was an abnormally long connection failure.

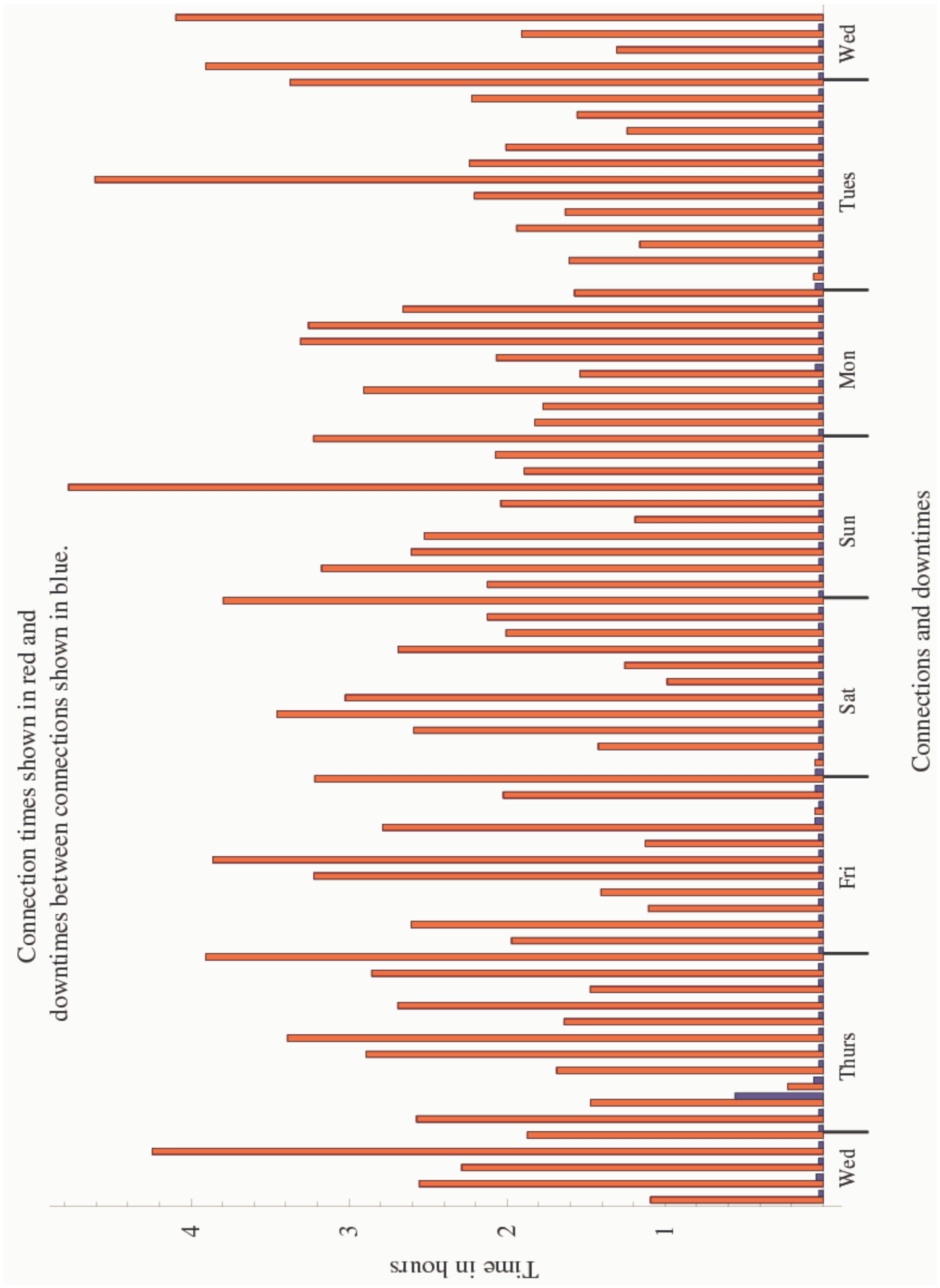

Figure 1: Total uptime and downtime in hours

.

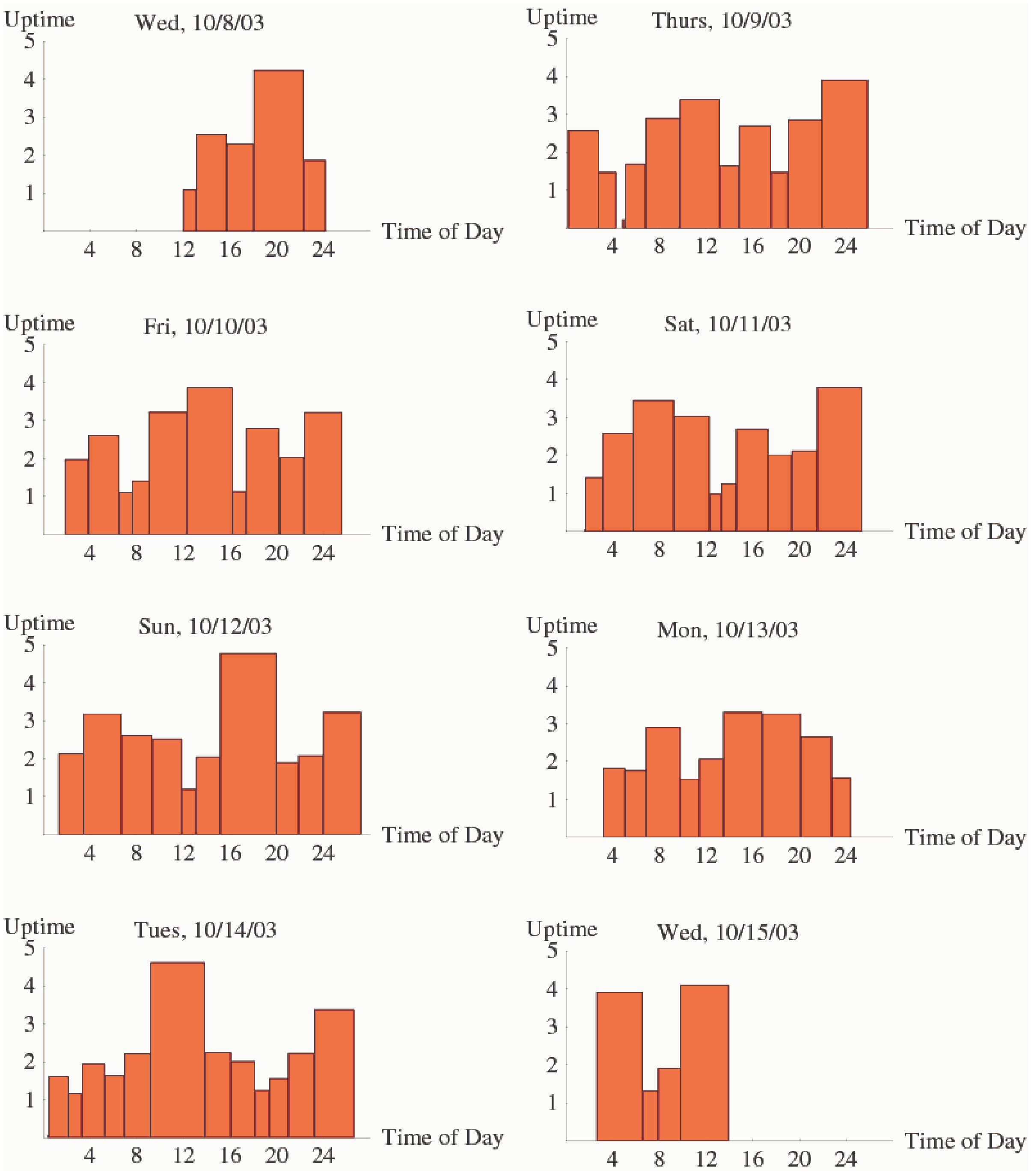

Figure 2: Total uptime in hours for each connection by day.

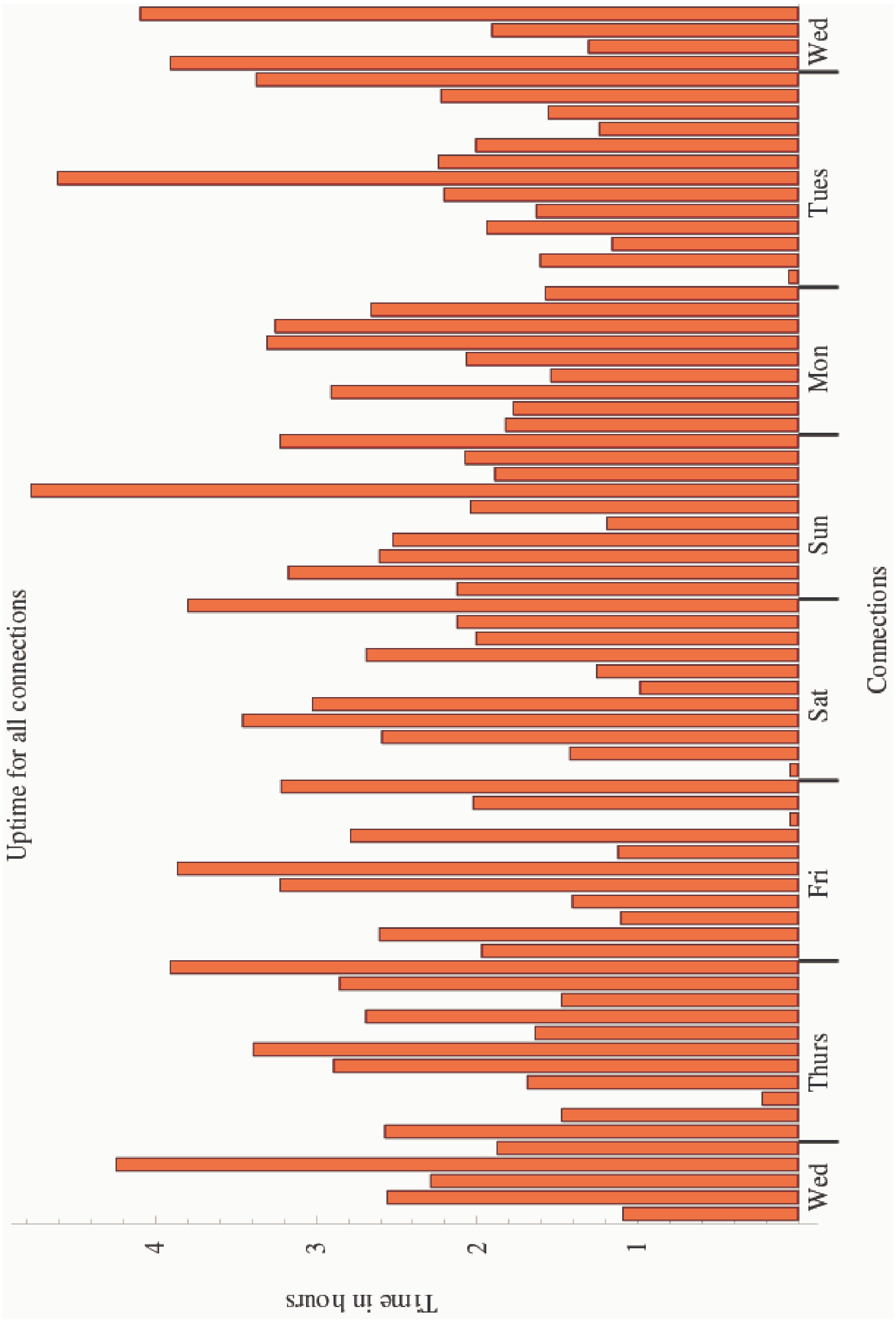

Figure 3: Uptime by hours for all connections.

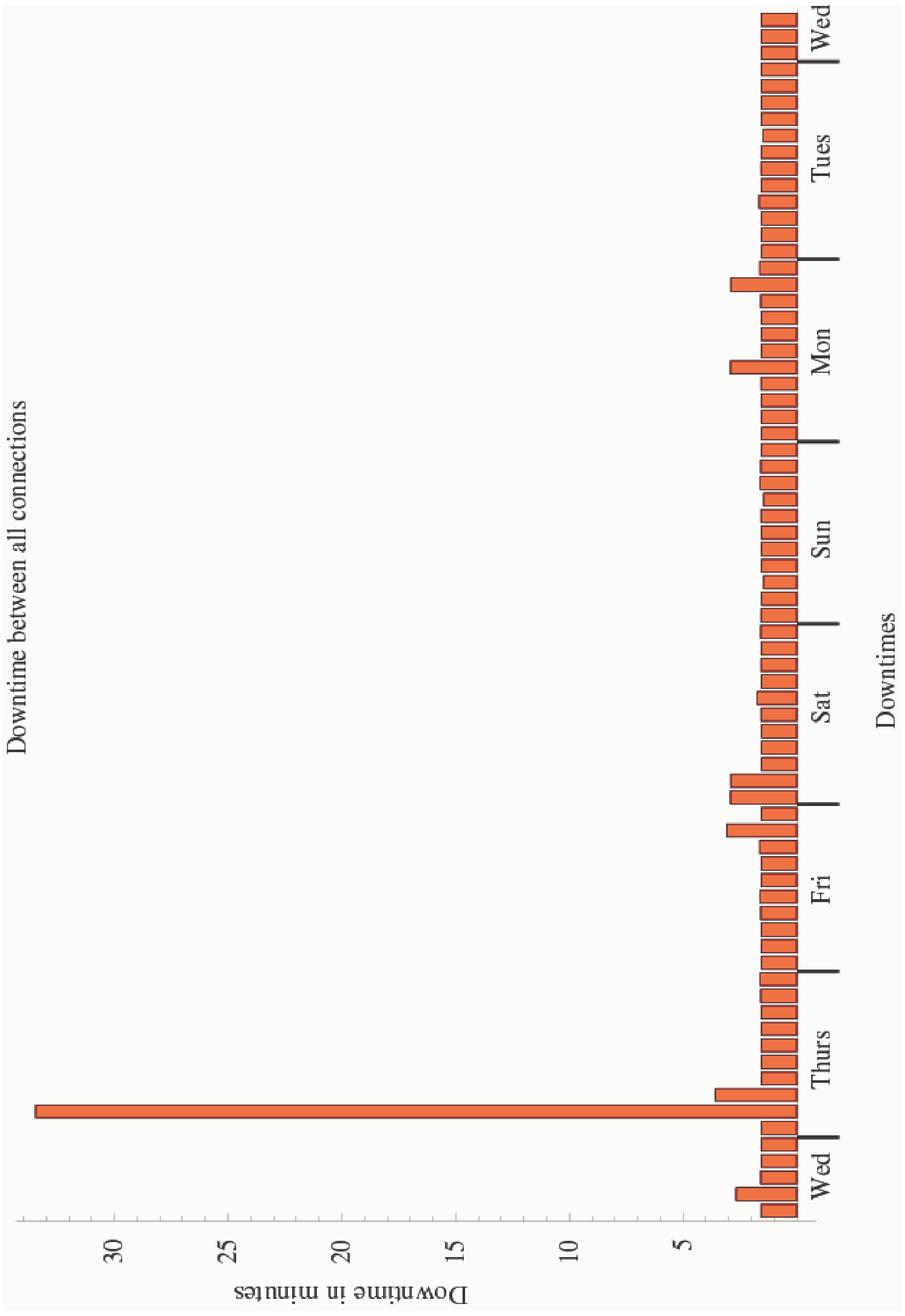

Figure 4: Downtime between connections in minutes.

3.4 Summary

It appears from the experiment that the sign functions as per the Hitech protocol documents and that communication between the server application and the sign worked as expected. Although the number of connection failures seems high at 73, the sign quickly re-establishes the connection to the server and has an overall connection uptime of 98%.

4. Brief Report on Hitech Sign Tests: January 27, 2004

Serial Numbers: CM04280003, CM04280004, CM04280005, CM04280009

The ITS/UW group monitored the connection characteristics of each of four Hitech signs connected to a central server application at ITS/UW. The monitoring period commenced at approximately 15:30 on Friday, January 23, 2004 and ended at approximately 13:00 on Mon, January 26, 2004. The signs are configured with the Online Timeout and the Offline Timeout set at 60 seconds. We plan to deploy the signs with these values set at 30 seconds, which is the minimum specified by Hitech.

Below are four figures, one for each sign, and each figure has two plots. There are three Sprint-serviced signs labeled 3, 4, 5, and one Verizon sign labeled 9. The top plot in each figure is the histogram of the session duration or time between reconnections. The mean session duration, in hours, is called out at the top and is labeled “Mean.” The mean time to reconnect is also at the top and labeled “Restart time.” The bottom plot in each figure has a bar that represents the time at which a reconnection occurred and the height of the bar represents the time it takes to reconnect. Note that when we deploy the signs with 30-second timeouts this reconnection time should be cut in half.

For each pair of figures, the percentage of downtime is show at the top right of the bottom plot. The downtime percentages range from 1.5 to 2.7 percent. The Verizon sign, number 9, has more reconnections but is faster to complete the reconnection. Though not shown in the graphs. the data indicates that the Verizon sign changes IP address each time a reconnection takes place.

During the testing, a software sign emulator was run in parallel and no reconnects were observed, indicating that the server process was continuously functional.

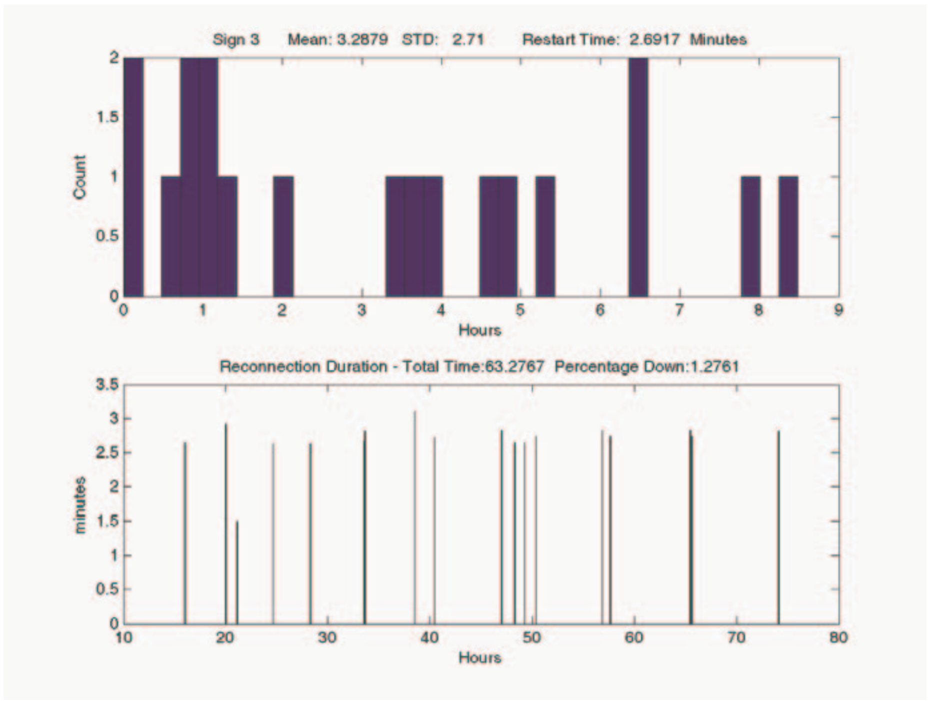

Figure 5: Sprint-serviced Sign 3 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

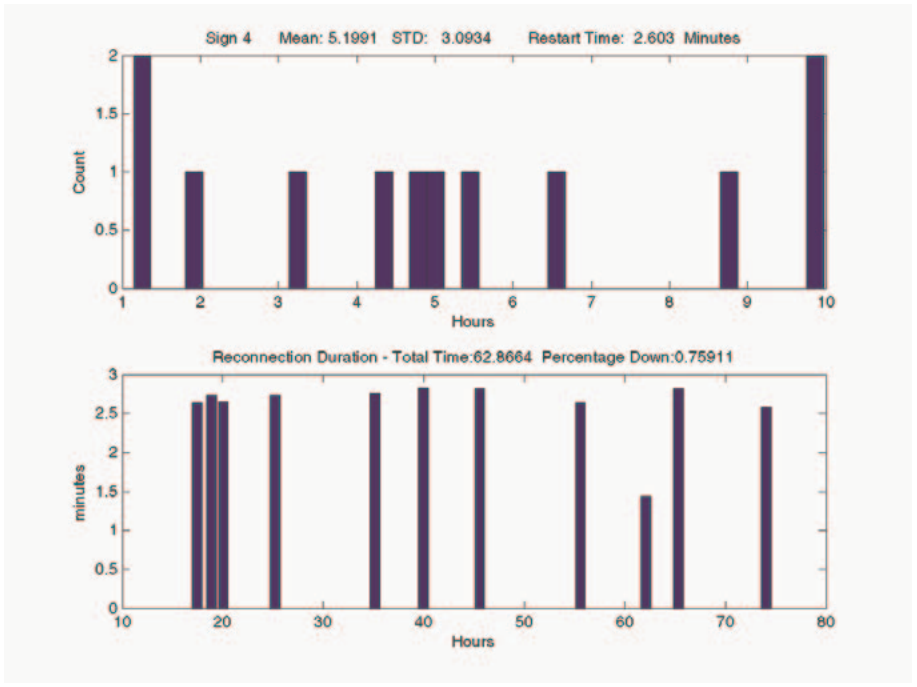

Figure 6: Sprint-serviced Sign 4 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

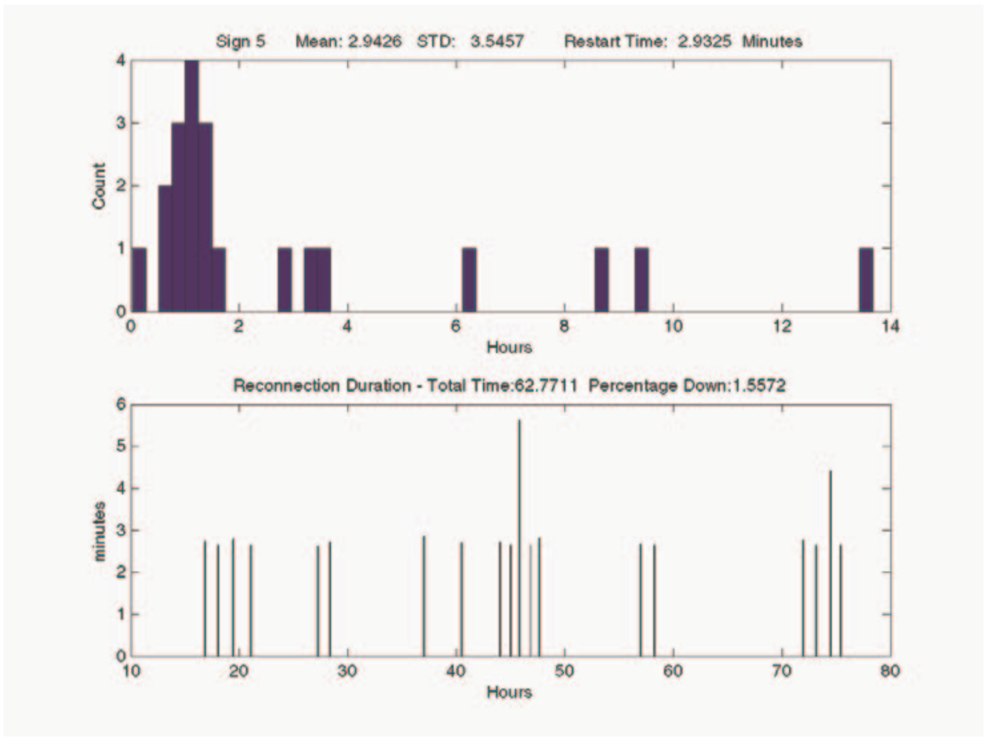

Figure 7: Sprint-serviced Sign 5 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

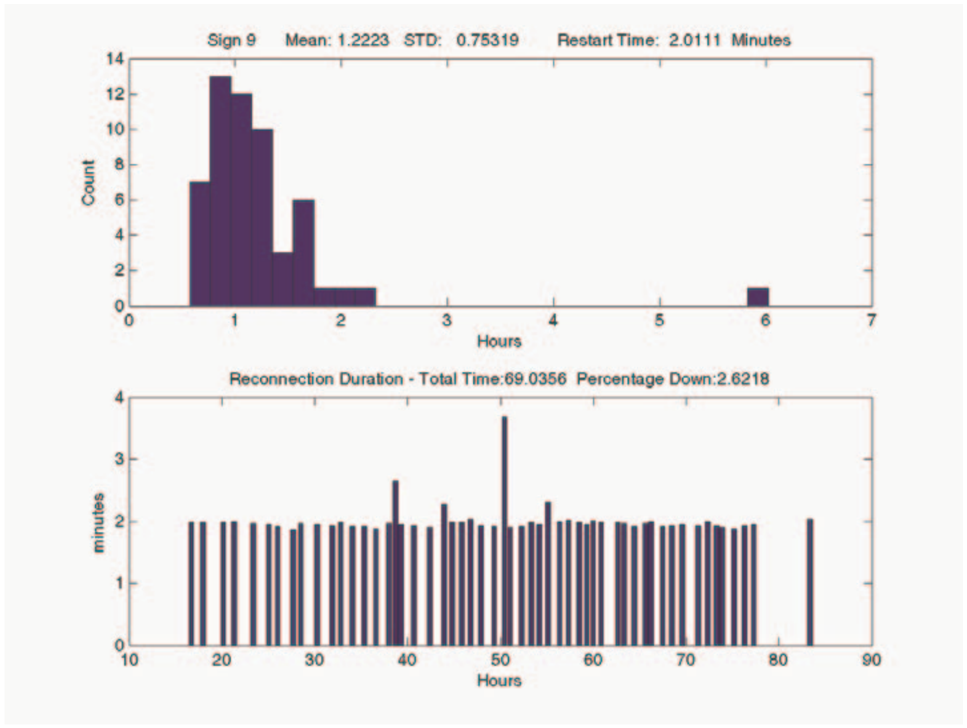

Figure 8: Verizon-serviced Sign 9 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

5. Brief Report on Hitech Sign Tests: May 18, 2004

Serial Numbers: CM04280003, CM04280004, CMO428005, CMO428006, CMO428008, CM04280009

The ITS/UW group monitored the connection characteristics of each of the Hitech signs connected to a central server application at ITS/UW. The monitoring period commenced at approximately 00:00 on Sunday, April 18, 2004 and ended at approximately 14:00 on Monday, May 17, 2004. The signs are configured with the Online Timeout and the Offline Timeout set at 30 seconds.

Below are twelve figures, two for each sign. There are four Sprint-serviced signs labeled 3, 4, 5, 6 and two Verizon-serviced signs labeled 8 & 9. The top plot for each sign is the histogram of the session duration or time between reconnections. The mean session duration, in hours, is called out at the top and is labeled “Mean.” The mean time to reconnect is also at the top and labeled “Restart time.” The bottom plot in each figure has a bar that represents the time at which a reconnection occurred, and the height of the bar represents the time it takes to reconnect.

For each pair of figures, the percentage of downtime is show at the top right of the bottom plot. The downtime percentages range from 0.97 to 2.4 percent.

Sign Information:

```
# serial number : location|ISP|ESN|Netphone|modem password

# initial 4 signs installed March 2004...
CM04280003 : Aurora IB @ 46th|Sprint|09900283353|206 251 6370|*******
CM04280004 : Aurora OB @ 46th|Sprint|09900283348|206 251 6860|*******
CMO428008 : Aurora IB @ 85th|Verizon|09900491255|206 714 7723|*******
CM04280009 : Aurora OB @ 85th|Verizon|09900481680|206 714 7726|*******

# sign 5 installed April 2004...
CMO428005 : Aurora OB @ 85th|Sprint|09900283340|206 251 6488|*******

# sign 6 installed (testing) May 2004...
CMO428006 : 46th IB @ Aurora|Sprint|09900147303|206 852 4091|*******
```

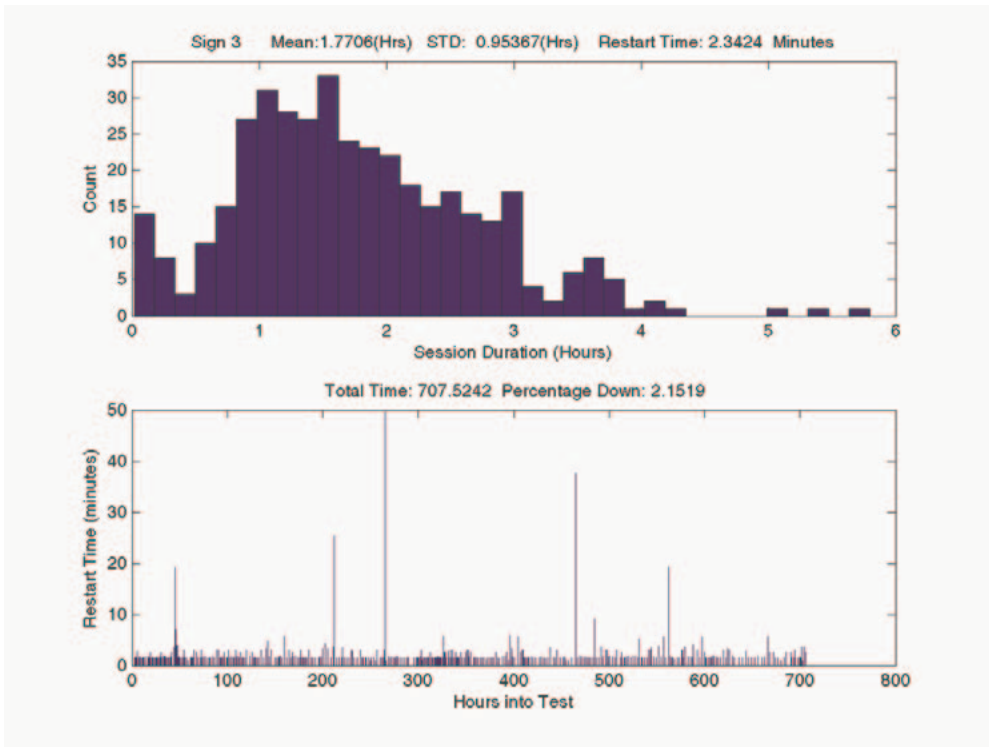

Figure 9: Sprint-serviced Sign 3 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

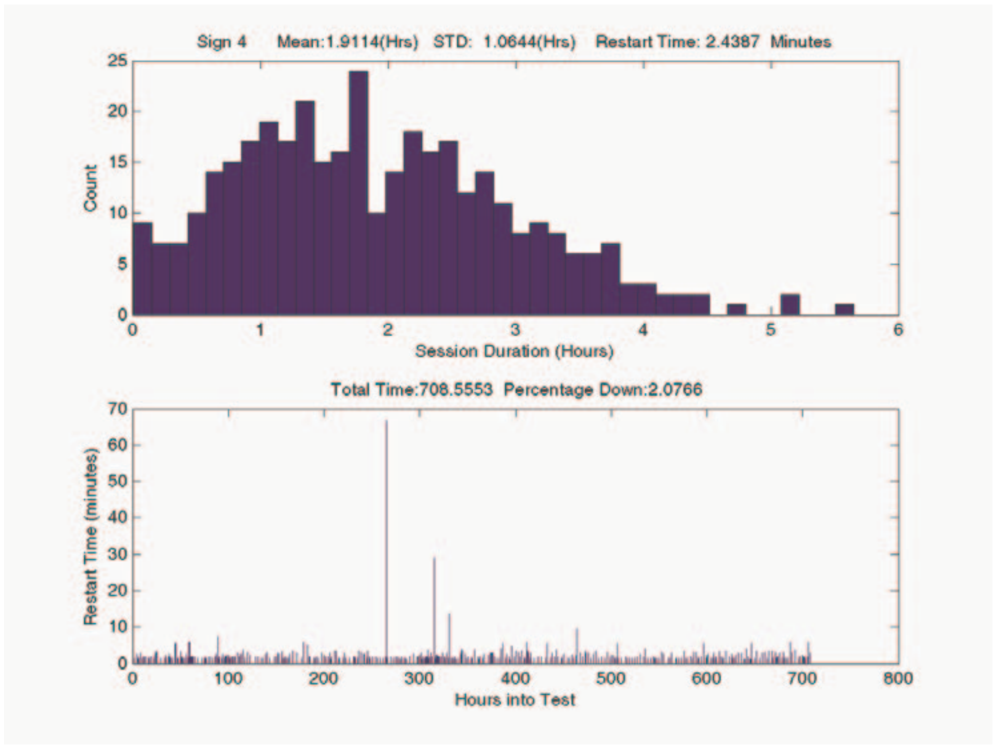

Figure 10: Sprint-serviced Sign 4 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

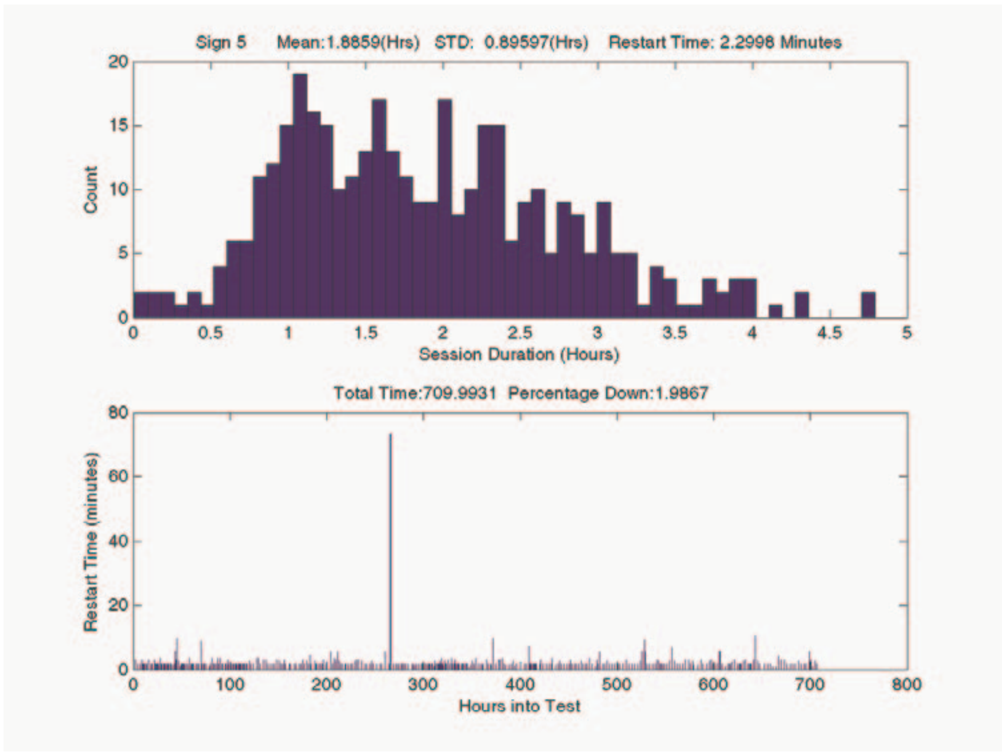

Figure 11: Sprint-serviced Sign 5 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

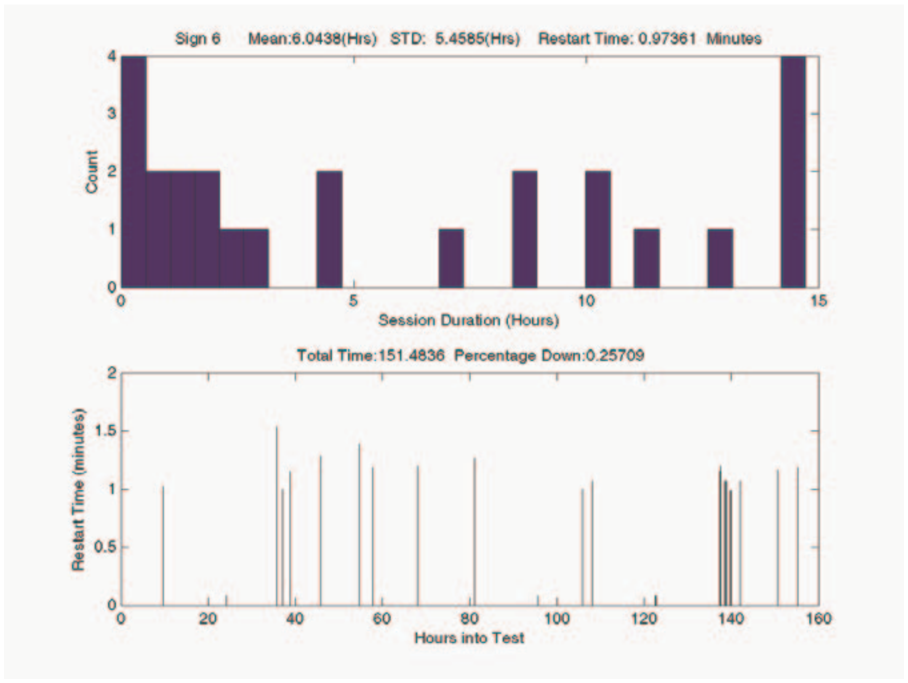

Figure 12: Sprint-serviced Sign 6 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect.

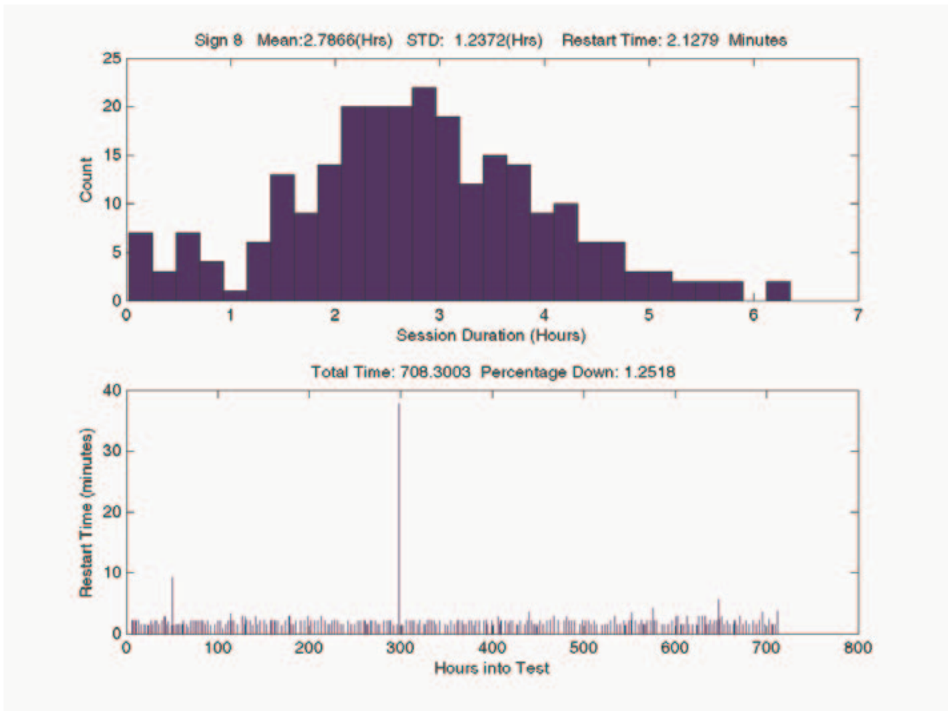

Figure 13: Verizon-serviced Sign 8 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect

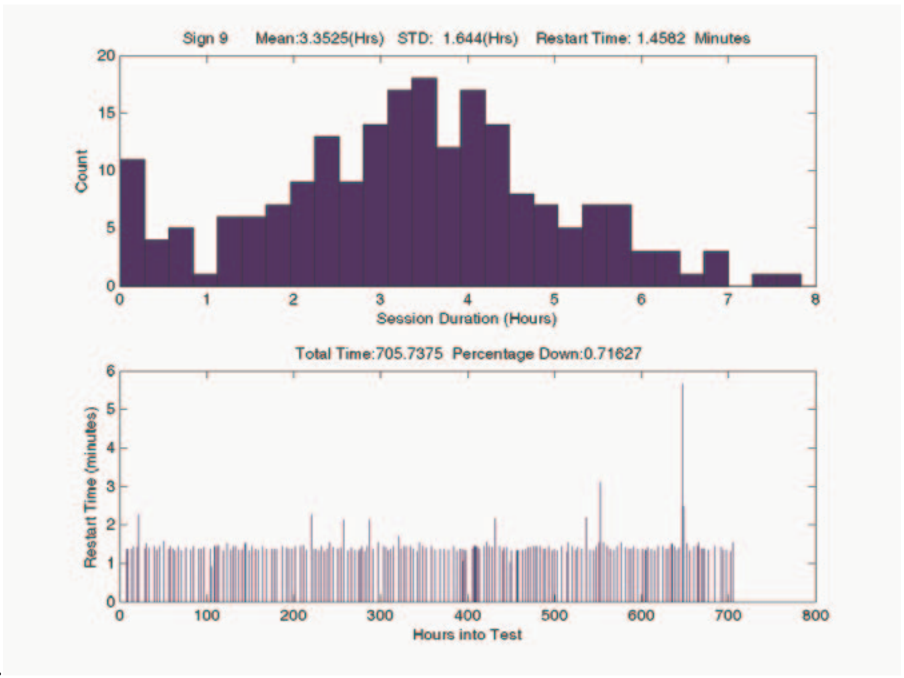

Figure 14: Verizon-serviced Sign 9 – top plot shows session duration or time between reconnections and bottom plot shows reconnection times and length of time to reconnect