ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ Η/Υ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ | ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Υλοποίηση ενός μηχανισμού για την αποτίμηση του πρωτοκόλλου SCTP κάτω από πραγματικές συνθήκες σε ασύρματα δίκτυα

ΒΑΣΙΛΗΣ ΜΠΟΥΡΛΙΑΚΑΣ

Επιβλέποντες:

Αθανάσιος Κοράκης
Αντώνιος Αργυρίου

ΒΟΛΟΣ 2014

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
SCHOOL OF ENGINEERING | UNIVERSITY OF THESSALY

Implementation of a framework for the evaluation of the SCTP protocol under real world environment settings over wireless networks

VASILIS BOURLIAKAS

Supervisors:

Athanasios Korakis
Antonios Argyriou

VOLOS 2014

This thesis was typeset using the LaTeX typesetting system.
The text is set 10/14pt on a 26pc body with Minion Pro.
Other fonts include Cronos Pro and Source Code Pro.

The overall layout of the text is inspired by the work of Edward Tufte, especially *Beautiful Evidence* (Tufte, 2006) and the guidelines found in *The Visual Display of Quantitative Information* (Tufte, 2001).

All graphics were produced using the PGF language via the Ti*k*Z and PGFplots packages.

Περίληψη

Ο σκοπός αυτής της διατριβής είναι να διερευνήσει και να ελέγξει την αποτελεσματικότητα του SCTP σε αντίθεση με το TCP υπό πραγματικές συνθήκες μέσα από μια σειρά πειραμάτων. Το SCTP είναι ένα σχετικά νέο πρωτόκολλο, που προτάθηκε ως διάδοχος του TCP, καθώς έχει σχεδιαστεί και κατασκευαστεί με βάση τις νέες ανάγκες και τις προδιαγραφές των σύγχρονων επικοινωνιών και του Διαδικτύου, αλλά μέχρι σήμερα δεν έχει βρει εφαρμογή σε μεγάλη κλίμακα. Για το λόγο αυτό, ο συγγραφέας αποφάσισε να δοκιμάσει περαιτέρω εφαρμογές, με στόχο την περιγραφή της καταλληλότητας της τεχνολογίας για περαιτέρω ανάπτυξη. Αυτή η διατριβή εξετάζει πρώτα τη διαδικασία ανάπτυξης της εφαρμογής και τις συνθήκες διεξαγωγής των πειραμάτων. Σε ένα δεύτερο στάδιο, πραγματοποιείται ένας σημαντικός αριθμός από πειράματα για τον έλεγχο της κύριας υπόθεσης. Τέλος, τα συγκεντρωμένα αποτελέσματα παρουσιάζονται και αναλύονται. Τα αποτελέσματα των πειραμάτων δείχνουν ότι το SCTP αποδεικνύεται ότι είναι καλύτερο στις περισσότερες από τις περιπτώσεις και ακόμα και κάτω από ιδανικές συνθήκες, ανταγωνίζεται επάξια το κατά πολύ παλιότερο και εξαιρετικά βελτιωμένο TCP.

Abstract

The aim of this thesis is to investigate and ascertain the efficiency of SCTP opposed to TCP under real world circumstances via a series of experiments. SCTP is a relatively new protocol, proposed as the successor of TCP as it is designed and built based on the emerging needs and specifications of modern communications and the Internet, but to date has found no large-scale application. For this reason the author decided to test further applications, with the aim of describing the technology's suitability for further development. This thesis first examines the development procedures followed and the experimentations setup. In a second stage, a significant number of experiments for testing the main assumption are undertaken. Finally, the pooled results are presented and analyzed. The results of the experiments show that SCTP proves to be better in most of the occasions and even on ideal scenarios, it competes well against the much older and extremely improved TCP.

Acknowledgments

I would like to express my deep gratitude to Dr. Athanasios Korakis, my head-supervisor, for welcoming me to the facilities of Network Implementation Testbed Laboratory of the Computer and Communication Engineering Department at University of Thessaly and providing me with the opportunity to conduct the experimental research of this thesis.

I would also like to thank Dr. Antonios Argyriou who agreed to undertake the supervision of my thesis.

My grateful thanks are also extended to Mr. Nikolaos Makris for his patient guidance, enthusiastic encouragement and useful critiques of this research work.

Special thanks to Mr. Spyridon Kechagias for his contribution and collaboration on the initial semester project on which the present thesis was based.

Finally, I wish to thank my family for their support, patience and encouragement throughout my study.

Bourliakas Vasilis,
Volos, 2014

Contents

1 Motivation

According to the OSI model[1], the transport layer provides the functional and procedural means of transferring variable-length data sequences from a source to a destination host via one or more networks, while maintaining the quality of service(QoS) functions. For networked applications, the choice of transport protocol is of major importance.

The most known and widely used, TCP, is byte-oriented, provides reliable data transfer and strict order-of-transmission delivery of data. The sense of a connection and a data stream are equivalent for TCP. Thus, many incompatibilities turn up among TCP and modern network applications, which requirements do not meet with the specifications of TCP. Either TCP does not provide the functionalities needed by the applications or results in overhead providing too many unnecessary ones.

HHTP is a message oriented protocol of the OSI application layer, where the most crucial requirement is a reliable and fast as possible transfer. For example while transferring multiple files, such as a website and all of its contents, the objective is to display everything as fast as possible to the user to provide an excellent QoS. The way TCP is designed ignores those factors, causing internet browsers to adapt for better results.

On the other hand SCTP is designed with a completely different logic. It is reliable, message oriented, thus not aiming on strict order transfers and also provides two great assets. The multiple stream feature, which allows multiple simultaneous file transfers and the multihoming feature, making SCTP efficient on more than one network interfaces. Considering the above facts, SCTP is assumed to be a more suitable and effective transport protocol for the HTTP traffic, in matters of reduced latency and greater throughput.

[1] The Open Systems Interconnection model (OSI) is a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers as in Table 1.1.

Table 1.1: OSI model by layer

Layer	Description
7.	*Application* — Network process to application
6.	*Presentation* — Data representation, encryption and decryption, convert machine dependent data to machine independent data.
5.	*Session* — Interhost communication, managing sessions between applications.
4.	*Transport* — Reliable delivery of packets between points on a network.
3.	*Network* — Addressing, routing and (not necessarily reliable) delivery of datagrams between points on a network.
2.	*Data link* — A reliable direct point-to-point data connection.
1.	*Physical* — A (not necessarily reliable) direct point-to-point data connection.

2 Stream Control Transmission Protocol

2.1 SCTP—How it works

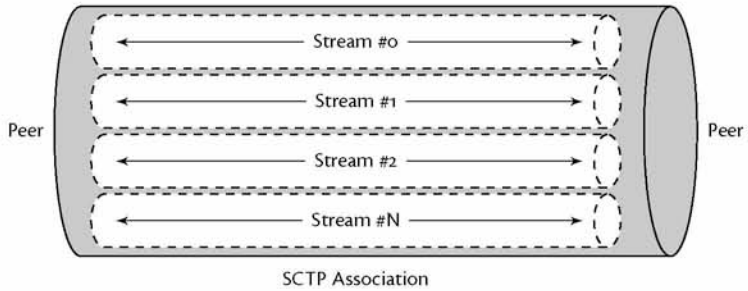
In computer networking, the Stream Control Transmission Protocol (SCTP) is a transport layer protocol, serving in a similar role to the popular protocols Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). It provides some of the same service features of both: it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP.

SCTP applications submit their data to be transmitted in messages (groups of bytes) to the SCTP transport layer. SCTP places messages and control information into separate chunks, each identified by a chunk header. The protocol can fragment a message into a number of data chunks, but each data chunk contains data from only one user message.

So, SCTP may be characterized as message-oriented, meaning it transports a sequence of messages, rather than transporting an unbroken stream of bytes as TCP does. As in UDP, in SCTP a sender sends a message in one operation, and that exact message is passed to the receiving application process in one operation.

2.2 Multi-streaming

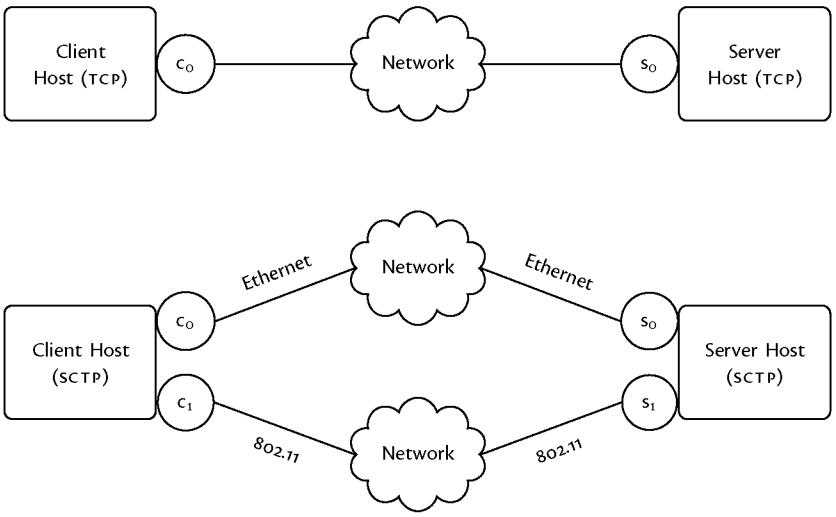
The term multi-streaming refers to the capability of SCTP to transmit several independent streams of chunks in parallel, for example transmitting web page images and the web page text simultaneously. In essence, it involves bundling several connections into a single SCTP association, operating on messages (or chunks) rather than bytes.



2.3 Multi-homing

A multi-homed host is one with more than one network interfaces, and therefore more than one IP addresses for which it can be addressed.

SCTP introduces the concept of an association that exists between two hosts but can potentially collaborate with multiple interfaces at each host.



2.4 *Determining interface*

One interface is established as the primary within STCP and all others become secondary. SCTP uses a built-in heartbeat to periodically check the status of the links and upon detecting a failure on the primary interface, the protocol sends traffic over the alternate path, without the applications being aware of or affected by that change.

2.5 *SCTP vs TCP*

As opposed to SCTP, the TCP has some major structural and functional differences. TCP supports single stream of data delivery whereas SCTP supports multiple streams of data delivery.

When it comes to data transfer, TCP is a byte-oriented protocol, while SCTP is message oriented.

Furthermore, one crucial difference of the two (2) protocols is the fact that SCTP supports multi-homing, as described above, whereas TCP requires a single TCP endpoint to have one IP address.

In addition, SCTP is considered to be a more secure protocol, since the ultimate objective of its development was to create a connection-oriented protocol better than TCP in speed, security, and redundancy.

3 Experimental Setup

3.1 Description

The main idea was the implementation of a simple HTTP server & client that serves static web pages over SCTP. Specifically, transferring a website, including the HTML file (index.html), an image file (nigel.jpg) and a video file (movie.mp4) over the SCTP server and comparing the obtained results to a corresponding conventional TCP HTTP server.

3.2 Experiment Environment

All experiments were conducted at the Network Implementation Testbed Laboratory of the Computer and Communication Engineering Department at University of Thessaly. NITLab has developed a testbed named NITOS, which stands for Network Implementation Testbed using Open Source platforms. The NITOS testbed currently consists of 50 operational wireless nodes, which are based on commercial Wi-Fi cards and Linux open source drivers. The testbed is designed to achieve reproducibility of experimentation, while also supporting evaluation of protocols and applications in real world settings.

3.3 Wi-Fi setup

Wi-Fi experimentation was conducted at the outdoor nodes of the NITOS testbed. The upgraded WiFi nodes have been developed by the NITLab team and support MIMO operation. The new version of outdoor nodes are equipped with 802.11a/b/g and 802.11a/b/g/n wireless interfaces. They also feature 2-core Intel cpus, new generation solid state drives and usb web cameras. Last but not least, each node is equipped with light, temperature and humidity sensors. For detailed specifications refer to Table 3.1

NITLab's Chassis Manager Card (CM card) is used to control and monitor Icarus node's operation. A tiny web server is running on the CM card and serves http requests, such as power on/off and reset commands. Furthermore, CM card can support real time sensor measurements, since it can be connected with temperature and humidity sensors as well as with light sensors. Two leds are

Table 3.1: Specifications

Motherboard	Features two Gigabit network interfaces and supports two wireless interfaces
CPU	CPU Intel Core 2 Duo P8400 2.26 GHz
RAM	2G DDR3
Wireless Interfaces	Atheros 802.11a/b/g & Atheros 802.11a/b/g/n (MIMO)
Chassis Manager card	NITlab CM card
Storage	Solid state drive
Power Supply	350 Watt mini-ATX
Antennas	Multi-band 5dbi , operates both on 2.4Ghz & 5Ghz
Pigtails	High quality pigtails (UFL to RP-SMA)

Figure 3.1: NITLab Outdoor Testbed

used to indicate the operation status of the outdoor nodes. More specifically a two state led, located on the side, indicates the power status of the node. If the led is red the node is turned off and if the led is blue the node is turned on. The same led flashes in red mode each time the cm card serves a request. The second led is on the CM card and indicates the connection of the node to the power supply.

Figure 3.2: An outdoor node

3.4 *WiMAX Setup*

The setup NITOS testbed is currently using is a fixed setup (employing no mobility between BSs) that does not require an ASN-GW installation. The overall topology is summarized in Figure 3.3.

For the proper configuration of the BS, the Netspan utility is required, provided by Airspan. Netspan is operating as an integrated service in a Windows Server 2008 box, and allows configuration of both the BS unit and the Service Flow on the WiMAX clients (Subscriber Stations – SSs). The current setup provides experimentation support with WiMAX enabled clients, using a fixed frequency. Apart from the Base Station's components, NITlab has also acquired several end-devices able to operate as WiMAX clients. This equipment is installed at NITOS testbed, enhancing existing nodes capabilities by adding 10 WiMAX enabled mini-pcie cards, 6 WiMAX USB dongles as well as 10 WiMAX enabled smartphones carried by volunteers. All of them use Open Source firmware/drivers, thus enabling more complex experiments, and the evolution of new MAC Layer standards. In addition to the aforementioned equipment, NITlab also acquired 3 indoor WiMAX to WiFi gateway units, enabling the creation of heterogeneous topologies by involving both WiMAX and WiFi technologies. More specific, the end-device used for the particular WiMAX experimentation was a Teltonika UM6225 USB dognle.The Teltonika UM6225 module is a compact USB modem that offers high speed data rate connectivity to WiMAX network. Key Features:

- Compliant with IEEE802.16e-2005 WiMAX Wave 2 standard
- USB 2.0 Interface

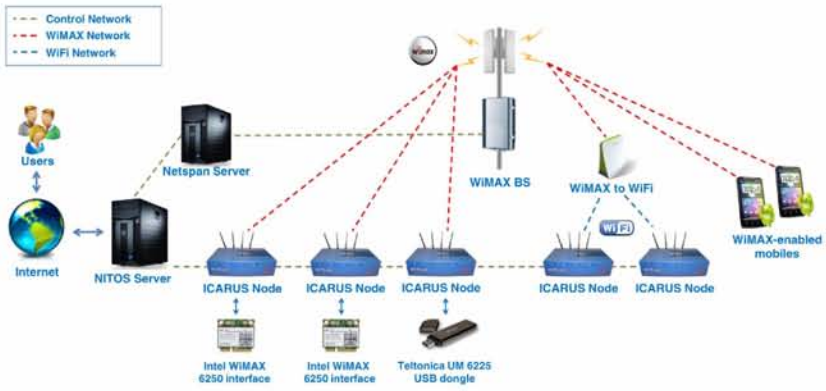


Figure 3.3: Topology of the testbed setup

- Operation within 2.3–2.4, 2.5–2.7, 3.3–3.6 or 3.3–3.8 GHz Range
- 2 Tx with Closed Loop Diversity
- Support up to HARQ category 7 (40 Mbps throughput DL+ UL)
- MIMO Matrix A and B for Improved Transmission Speed and Coverage
- Compatibility with Windows XP/Vista/7, Linux, MAC OS X

Figure 3.4: Icarus node

3.5 *Experiment specifications*

The main setup on all experiments was identical, two (2) nodes, one serving as the server and the other as the client. Initially, both the TCP and SCTP server code were simultaneously executed on the server node and remained active until the end of all measurements, waiting for requests. On the client node, TCP and SCTP client code were executed alternately, as close as possible in matters of time so that the state of the network was almost identical for both measurements.

Starting with The Wi-Fi setup, initial measurements were carried out on the 802.11g mode. 802.11g is the third modulation standard for wireless LANs. It works in the 2.4 GHz band and operates at a maximum raw data rate of 54 Mbit/s. Despite its major acceptance, 802.11g suffers from interferences in the already crowded 2.4 GHz range. Devices operating in this range include microwave ovens, Bluetooth devices, baby monitors and digital cordless telephones, which can lead to interference issues. Additionally, the success of the standard has caused usage/density problems related to crowding in urban areas.

Next up, the Wi-Fi experimentation setup was switched to the 802.11a mode. 802.11a provides protocols that allow transmission and reception of data at rates of 1.5 to 54 Mbit/s. It has seen widespread worldwide implementation, particularly within the corporate workspace. The 802.11a standard uses the same data link layer protocol and frame format as the original standard, but an OFDM based air interface (physical layer), which yields realistic net achievable throughput. It operates in the 5 GHz band with a maximum net data rate of 54 Mbit/s, plus error correction code. Since the 2.4 GHz band is heavily used to the point of being crowded, using the relatively unused 5 GHz band gives 802.11a a significant advantage. It also suffers from interference, but locally there may be fewer signals to interfere with, resulting in less interference and better throughput.

Table 3.2: Measurements specifications

	Wi-Fi Protocol 802.11g				Wi-Fi Protocol 802.11a			
Rates (Mbps)	6	9	12	18	6	9	12	18
Transmisson power	0–27	0–27	0–27	0–27	default	default	default	default
Rates (Mbps)	24	36	48	54	24	36	48	54
Transmisson power	0–27	0–27	0–27	0–27	default	default	default	default

	WiMAX Uplink & Downlink		
MCS	16 QAM 1/2	16 QAM 3/4	64 QAM 1/2
MCS	64 QAM 2/3	64 QAM 3/4	64 QAM 5/6
MCS	QPSK 1/2	QPSK 3/4	default

Finally, the whole experiment was set up on a WiMAX network. As a standard intended to satisfy needs of next-generation data networks (4G), WiMAX is distinguished by its dynamic burst algorithm modulation adaptive to the physical environment the RF signal travels through. Modulation is chosen to be more spectrally efficient. There is no uniform global licensed spectrum for WiMAX, however the WiMAX Forum has published three licensed spectrum profiles: 2.3 GHz, 2.5 GHz and 3.5 GHz, in an effort to drive standardisation and decrease cost. Comparisons and confusion between WiMAX and Wi-Fi are frequent, because both are related to wireless connectivity and Internet access.

- WiMAX is a long range system, covering many kilometres, that uses licensed or unlicensed spectrum to deliver connection to a network, in most cases the Internet.
- Wi-Fi uses unlicensed spectrum to provide access to a local network.
- Wi-Fi is more popular in end user devices
- Wi-Fi runs on the Media Access Control's CSMA/CA protocol, which is connectionless and contention based, whereas WiMAX runs a connection-oriented MAC.
- WiMAX and Wi-Fi have quite different quality of service (QoS) mechanisms:

  WiMAX uses a QoS mechanism based on connections between the base station and the user device. Each connection is based on specific scheduling algorithms. Wi-Fi uses contention access — all subscriber stations that wish to pass data through a wireless access point (AP) are competing for the AP's attention on a random interrupt basis. This can cause subscriber stations distant from the AP to be repeatedly interrupted by closer stations, greatly reducing their throughput.

The exact specifications and variables of all measurements conducted in the context of this thesis are summarized in Table 3.2

4 Implementation

4.1 SCTP Web Server/Client Implementation

The application was mostly implemented in C, with minor C++ code additions for handling exceptions.

Supplementary tools used for the SCTP sockets programming were the `lksctp-tools` (Linux Kernel Stream Control Transmission Protocol Tools) and the corresponding C language library called `libsctp`, which provide certain functions for implementing SCTP protocol functionality.

The development of the SCTP Web Server/Client was originally based on an existing implementation of a TCP Web Server/Client which was altered in such a way that it could support the SCTP protocol. Most significant additions made are listed below:

- The requested HTML file is initially parsed on the server side to ascertain which and what kind of files should be transferred.
- As mentioned above, socket functions were modified to fit the SCTP functionality, based on the `libsctp` library.
- A simple allocation/reallocation strategy for the different file types was implemented and added to the original application in a way that HTML files got the highest priority, followed by the image and video files accordingly.
- The file chunk assembly policy was modified in order to work based on the stream the chunk originated from.
- Parts of C++ code were added to the original implementation for the purpose of exception handling during sending and receiving messages (`sctp_sendmsg`/`sctp_recv_msg`).

4.2 HTML Parsing

After the client's request for the HTML file, the server application begins parsing the above file, looking for patterns corresponding to image (`*.jpg`) and video (`*.mp4`) files. Thereafter, those files are passed onto a "sending" list where the different streams are allocated to each file, according to its type.

A more analytical description of the stream allocation strategy follows up on a later section.

4.3 SCTP Network Programming

SCTP socket programming bears many similarities to the TCP one, but with a few inevitable modifications. More specifically, to fully exploit the capabilities and functionality provided by SCTP (such as the use of multiple streams), it is necessary to use some new functions and data structures provided by `libsctp`. Examples of such code fragments are given below:

```
TCP:   sockfd = socket(AF_INET, SOCK_STREAM, 0);
SCTP:  sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);

/* Enable receipt of SCTP Snd/Rcv Data via sctp_recvmsg */
setsockopt(sockfd, SOL_SCTP, SCTP_EVENTS, (const void *)&events,
sizeof(events));
```

↑ Input

↓ Input

4.4 Initial Stream Allocation Strategy

The implemented application uses a quite simple stream allocation strategy in order to ensure that the highest priority is granted to the HTML file. The files about to be sent are stored in a list, which is later ordered according to the file types, with increasing priority, as shown in Figure 4.1.

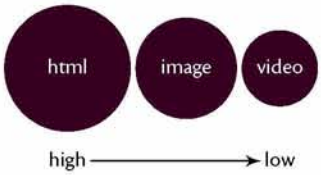


Figure 4.1: File type priorities

Subsequently, a stream is assigned to each file selected for sending. In this implementation the maximum number of files that can be transmitted simultaneously, ie to share the available streams, is set to three (3).

- If the HTML file exists in the "sending" list, all available streams are allocated to it

- Otherwise, the available streams are equally assigned to the rest of the files, one for each file sequentially, as found on the "sending" list.

4.5 Stream Reallocation Strategy

After the completion of a file transfer, the streams used to transfer the above files remained inactive and the rest of the files continued using only the streams initially assigned to them. In this way, the multistreaming capabilities of the SCTP protocol were not fully exploited.

In order to fix that behavior, the inactive streams are now reallocated to the remaining files to be sent, based on the priority of each file. The client application is notified about the changes and the stream allocation is adjusted accordingly.

4.6 SCTP Multistreaming File Transfer

The original SCTP implementation is providing the correct sequence of messages per stream, with ID assignment to each data packet, but cannot ensure the proper sequencing of messages belonging to different streams (see Figure 4.2: if 2 data chunks are sent through different streams, there is no way to make a correlation between them).

When a file is moved through many different streams (as is done in our implementation), this issue should be properly addressed to ensure the correct assembly of the file on the client application. To solve the problem on the server side, a unique header-ID which holds the serial number of the chunk is injected into each chunk of a single file. On the client side, those global ID's are extracted and used to correctly assemble the files coming from different streams.

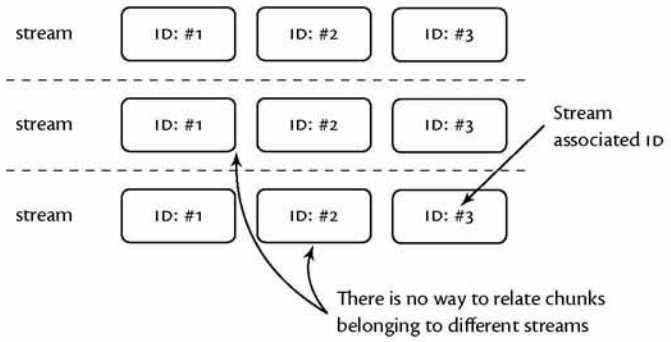


Figure 4.2: Chunk sequencing

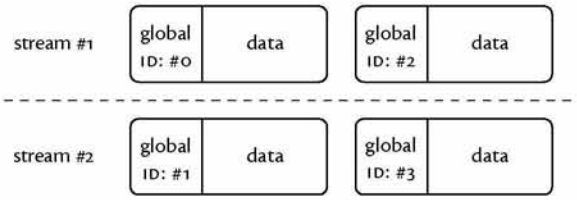


Figure 4.3: Data chunks with assigned header IDs

4.7 *File Reassembly On The Client Side*

Once the client has received the data chunks of a unique file in correct order, those are written directly on a binary file. In case some data chunks are received out of sequence, they are stored into a list until all the previous missing data chunks arrive, and then are written all together into the binary files in the proper sequence (see Figure 4.4 for an example).

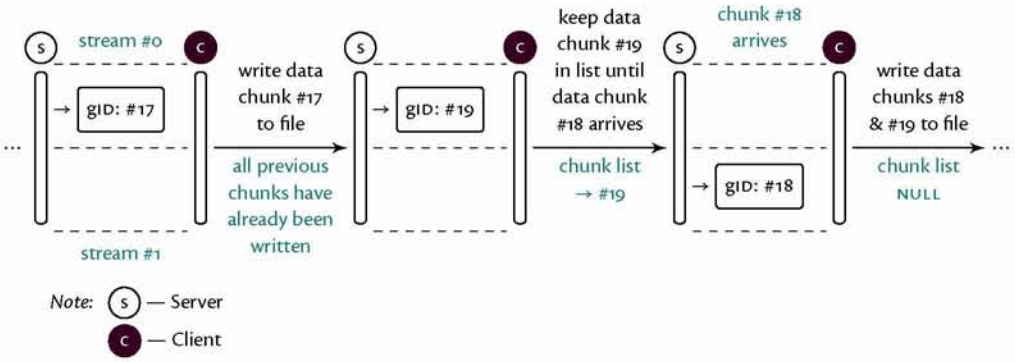


Figure 4.4: File tranfer through two different streams

4.8 *Exception Handling*

The system buffer for WIMAX in Linux drains on a regular basis which causes the file transfers to fail and the whole application to terminate, because of the server/client attempting to read data from an empty buffer. To avoid this phenomenon, a simple "try-catch" piece of C++ code was added for the exception handling on both the server and the client side.

```
try { sctp_sendmsg( fd, bufferSCTP, strlen(bufferSCTP), NULL, 0, 0, 0,
 curr_stream, 0, 0 ); }
catch (int e) {
```

↑ Input

```
cout << "An exception occurred. Exception Nr. " << e << '\n'; }
try { in = sctp_recvmsg(sockfd, bufferSCTP, MAXSIZE, NULL, 0, &sndrcvinfo,
 &flags); }
catch (int e) {
cout << "An exception occurred. Exception Nr. " << e << '\n'; }
```

Input

4.9 File Structs

In order to achieve the proper and complete transfer of the files, C structs were used to store the needed information.

Struct for the files sent from the server:

```
struct file {
    char filename[1024];
    char extension[5];
    char filetype[10];
    int curr_chunk_global_id;
    short int streams_assigned[OUT_STREAMS];  // 1 for yes, 0 for no
} *files[100];
```

Input

Struct for the files received from the client:

```
struct file {
    char filename[1024];
    char extension[5];
    char filetype[10];
    int curr_chunk_global_id;
    int last_chunk_written_id;
    short int streams_assigned[IN_STREAMS];  // 1 for yes, 0 for no
    struct chunklist *root; // points to the root of the list that keeps
    // chunks that arrived out of order
};
```

Input

4.10 TCP server & client

The TCP server & client implementation was more straightforward:

- After a client HTTP request, the server sends the files serially.
- Problem: TCP is byte stream-oriented, so `send()` function does not guarantee that all bytes sent out will actually be transferred in one batch.
- Solution: manually make sure that all bytes are sent – `sendall()` function.

```
int sendall(int s, char *buf, int *len) {
    int total = 0;         // how many bytes we've sent
    int bytesleft = *len;  // how many bytes we have left to send
    int n;

    while(total < *len) {
        n = send(s, buf+total, bytesleft, 0);
        if(n==-1) { break; }
        total += n;
    bytesleft -= n;
    }

    *len = total        // return number actually sent here
    return n==-1?-1:0;  // return -1 on failure, 0 on success
}
```

Input

4.11 OML Usage

Initially, OML parameters had to be added to the server | client code, set up and modified according to the purposes of this experiment. The necessary files were originally produced with two (2) simple commands executed on the terminal:

```
oml2-scaffold --app server
# produces server.rb (or client.rb)
oml2-scaffold --oml server.rb
# produces server_oml.h containing MP structs and functions
```

The corresponding headers and OML functions (initialize, add, start inject) were included to server.c and client.c. More specifically, the retrieved measurements were forwarded and saved on the OML server through a custom function called `inject_time()`, which can be seen below, along with a fragment of the OML usage on the server.

```
inject_time()
void inject_time(oml_mps_t *g_oml_mps, int *total_chunk_seq_num,
   int  *file_chunk_seq_num,
   char *fname, int stream, uint64_t time_ns,
   uint64_t time_diff, int when, char *mode) {
    char total_chunk_seq_str[1024];
    char file_chunk_seq_str[1024];

    sprintf(total_chunk_seq_str, "chunk#%d",
    *total_chunk_seq_num); // build e.g. "chunk#1302"
    if(filename != NULL) strcpy(file_chunk_seq_str, fname);
 // e.g. filename = "HTML_HEADER"
    // build e.g. "movie.mp4#213"
    else { sprintf(file_chunk_seq_str, "FILE%d", *file_chunk_seq_num); }

    // inject TIME DIFFERENCE
    if(!strcmp(mode,"TIME_DIFF"))
        oml_inject_timeDiff(g_oml_mps->timeDiff, total_chunk_seq_str,
        file_chunk_seq_str,
              fname, stream, time_diff);

    // inject TIME POINT
    else if(!strcmp(mode,"TIME_POINT")) {
        oml_inject_timePoint(g_oml_mps->timePoint,total_chunk_seq_str,
        file_chunk_seq_str,
             fname, stream, time_ns);
        // if the measurement is sent BEFORE sctp_sendmsg,
 increment the chunk seq. numbers
        if(when == BEFORE_SEND) {
            (*total_chunk_seq_num)++;
            (*file_chunk_seq_num)++;
        }
    }
}
```

4.12 Measurements example

```
curr_time = prev_time = ClockGetTime();
// get time BEFORE sctp_sendmsg
inject_time(g_oml_mps, &total_chunk_seq_num, &file_chunk_seq_num,

 files[sq[fq_ind]->file_idx]->filename, curr_stream,
 curr_time, 0, BEFORE_SEND, "TIME_POINT");

try { sctp_sendmsg( fd, bufferSCTP,
 (size_t)(bytes_read + CHUNK_SN_BYTES + 1),
NULL,0,0,0,
   curr_stream,0,0 ); }
catch (int e) {
cout << "An exception occurred. Exception Nr. " << e << 'n'; }

curr_time = ClockGetTime();      // get time AFTER sctp_sendmsg
inject_time(g_oml_mps, &total_chunk_seq_num, &file_chunk_seq_num,
files[sq[fq_ind]->file_idx]->filename,curr_stream, curr_time,
0, AFTER_SEND, "TIME_POINT");
inject_time(g_oml_mps, &total_chunk_seq_num, &file_chunk_seq_num,

    files[sq[fq_ind]- >file_idx]->filename,
    curr_stream,0,curr_time - prev_time,0,TIME_DIFF");
```

Input

Input

4.13 OML Measurement Points (MPs)

Two (2) OML measurement points were defined in order to receive the required measurements:

- `timePoint`: stores a specific time using as reference another `timePoint`
- `timeDiff`: stores a certain amount of time as `timePoint1 - timePoint2`

Below is the structure (C struct) automatically generated by the `oml2-scaffold` containing the OML measurement points:

```
typedef struct {
    OmlMP* timePoint;
    OmlMP* timeDiff;
    OmlMP* totalGoodput;
} oml_mps_t;
```

Input

Input

5 Results

Referring back to Table 3.2, the overall number of measurements carried out in this thesis exceeds 550. A thorough presentation and analysis of the results obtained, divided into three (3) major sections (Wi-Fi 802.11g, Wi-Fi 802.11a and WiMAX), follows on this chapter. Finally, as a reminder, the files transmitted on each experiment and their sizes are listed on Table 5.1

Table 5.1: File types and sizes.

File	Size
INDEX.HTML	333 B
NIGEL.JPG	9.94 KB
MOVIE.MP4	311 KB

5.1 Wi-Fi 802.11g Results

As the most common and widely used Wi-Fi mode, 802.11g, was inquired in greater depth than the other experimental sections. With a total of 512 measurements conducted on all supported rates (6, 9, 12, 18, 24, 36, 48, 54), the transmission power varying from 0–27 on each rate and each experiment on this section repeated twice for greater accuracy, all possible scenarios were covered. The results shown on Table 5.2 are referring to the mean time difference of all measurements, regardless transmission power, grouped by rate. More specifically, we note that "index.html" is transmitted faster via SCTP across all rates, but with hardly noticeable differences. When it comes to "nigel.jpg", results are very close, with SCTP proving to be slightly faster in general, with minor exceptions at rates 18 and 48 where TCP is faster. A significant occurrence is noticed at rate 56, where the difference between the two (2) protocols is huge (1 second). Eventually, on "movie.mp4" which is the largest file, results are divergent. TCP is steadily faster with a major exception at rate 56, where SCTP is faster with a vast difference of 5.1 seconds.

5.2 Wi-Fi 802.11a Results

The number of measurements carried out on the 802.11a mode is nowhere close to the corresponding on 802.11g. The reason is simple, the state of the network over 802.11a mode is close to ideal, with minor exterior interference as explained on chapter 3, so no major variations were expected. Again, as on the 802.11g mode, the results shown on Table 5.3 refer to the mean time difference of all

	6			9			12			18		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0008	0.0018		0.0010	0.0017		0.0007	0.0023		0.0008	0.0017	
NIGEL.JPG	0.0235	0.0268		0.0168	0.0161		0.0138	0.0155		0.0428	0.0194	
MOVIE.MP4	1.0586	0.9547		0.7463	0.6352		0.5438	0.4075		0.6144	0.5165	

	24			36			48			54		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0007	0.0009		0.0005	0.0010		0.0006	0.0015		0.0004	0.0010	
NIGEL.JPG	0.0096	0.0099		0.0074	0.0082		0.0429	0.0131		0.1054	1.1573	
MOVIE.MP4	0.3380	0.3055		0.3124	0.1896		0.4299	0.3440		0.4808	5.5306	

Table 5.2: Mean Time Difference (in seconds) across all measurements and transmission powers for the Wi-Fi Protocol 802.11g

	6			9			12			18		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0005	0.0013		0.0004	0.0008		0.0010	0.0009		0.0008	0.0006	
NIGEL.JPG	0.0171	0.0205		0.0122	0.0117		0.0096	0.0113		0.0069	0.0072	
MOVIE.MP4	0.7682	0.5140		0.6053	0.3597		0.3264	0.2745		0.2387	0.1906	

	24			36			48			54		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0009	0.0005		0.0009	0.0006		0.0008	0.0007		0.0008	0.0008	
NIGEL.JPG	0.0058	0.0054		0.0042	0.0041		0.0042	0.0031		0.0041	0.0028	
MOVIE.MP4	0.2025	0.1591		0.1580	0.1176		0.1390	0.0956		0.1299	0.0884	

Table 5.3: Mean Time Difference (in seconds) across all measurements for the Wi-Fi Protocol 802.11a

measurements grouped by rate. Results were quite different from above. TCP seems to prevail on all rates and file types, but differentiation is minor on the majority of cases. The only noticeable differences are noticed on the largest file ("movie.mp4") transfer.

As observed in section 5.2, TCP seems to have a slightly better performance on the ideal environment provided on Wi-Fi 802.11a. Same thing is noticed on some distinct cases in section 5.1. There is a reasonable explanation for this behavior. TCP is a much older protocol, widely used in real life applications and therefore its implementation on the Linux Kernel is significantly improved over and over throughout all these years. Thus, TCP is gaining a slight advantage over SCTP on those marginal situations.

5.3 *WiMAX Results*

A different separation was used for the WiMAX experimentation. Initially, the downlink of the network was measured. Pertaining to computer networks, a downlink is a connection from data communications equipment towards data terminal equipment (real server serving as the server application and the client node as the client application). Thereafter, the uplink was measured with reverse topology. Pertaining to computer networks, an uplink is a connection from data communications equipment toward the network core (real server serving as the client application and the client node as the server application). In consistency with the previous policy, the results refer to the mean time difference of all measurements, grouped by modulation scheme (MCS).

5.4 *WiMAX Downlink Results*

For the first time, very divergent results were noticed here. SCTP proves to be much faster on the HTML file ("index.html") transfer across all modulations schemes. Contrariwise, on the image ("nigel.jpg") transfer, TCP had great performance, leaving SCTP way behind in matters of time on all occasions. Finally, during the video transfer ("movie.mp4"), SCTP was once again faster than its competitor on most cases. TCP was more agile at the 16QAM ½ and the QPSK ½ MCS, with noticeable difference. For further reference, see Table 5.4.

	16 QAM 1/2			16 QAM 3/4			64 QAM 1/2			64 QAM 2/3		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0009	0.0813		0.0005	0.0763		0.0005	0.0815		0.0005	0.0808	
NIGEL.JPG	0.1591	0.0060		0.0739	0.0095		0.0848	0.0087		0.0857	0.0094	
MOVIE.MP4	1.6237	0.7591		0.8457	0.8909		0.8685	0.9590		0.8727	0.9240	

	64 QAM 3/4			64 QAM 5/6			QPSK 1/2			QPSK 3/4			default		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0004	0.0759		0.0005	0.0815		0.0002	0.0814		0.0005	0.0862		0.0005	0.0814	
NIGEL.JPG	0.0701	0.0087		0.0886	0.0057		0.0898	0.0096		0.0840	0.0093		0.0767	0.0092	
MOVIE.MP4	0.7940	0.9720		0.8147	0.9138		0.8393	0.5479		0.9000	0.7252		0.8173	0.9461	

Table 5.4: Mean Time Difference (in seconds) across all measurements for the WiMAX downlink

	16 QAM 1/2			16 QAM 3/4			64 QAM 1/2			64 QAM 2/3		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0006	0.0764		0.0003	0.0723		0.0003	0.0773		0.0004	0.0873	
NIGEL.JPG	0.0744	0.0870		0.0740	0.0693		0.0723	0.0793		0.0988	0.0693	
MOVIE.MP4	1.2403	4.5833		3.0988	3.5331		0.9925	3.7047		1.9752	2.7303	

	64 QAM 3/4			64 QAM 5/6			QPSK 1/2			QPSK 3/4			default		
File	SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP		SCTP	TCP	
INDEX.HTML	0.0004	0.0009		0.0003	0.0824		0.0008	0.0774		0.0003	0.0724		0.0004	0.0874	
NIGEL.JPG	0.0582	0.1591		0.0777	0.0870		0.0627	0.0842		0.0572	0.0787		0.1358	0.0740	
MOVIE.MP4	3.3261	1.6237		2.0311	0.7815		0.8204	1.5393		2.9203	1.3374		2.4538	1.4371	

Table 5.5: Mean Time Difference (in seconds) across all measurements for the WiMAX uplink

5.5 *WiMAX Uplink Results*

Afresh, a variety of results were noticed in this case and shown on Table 5.5. Overall, SCTP seems to overrule TCP, with a few exceptions. Across all MCS, SCTP performed better on the HTML and the image file transfer, excluding only three (3) cases for the image file (16QAM ¾ , 64QAM ⅔ and the default MCS). In cocnlusion, obtained results were balanced on the video file transfer. SCTP was faster on five (5) occasions (16QAM ½, 16QAM ¾, 64QAM ½, 64QAM ⅔ and QPSK ½) with differences varying from 0.5 to 3 seconds. On the other hand TCP achieved better times in the rest of the cases, with steady differences around 1 second. A small sample of plots obtained from the measurements are listed on the next page.

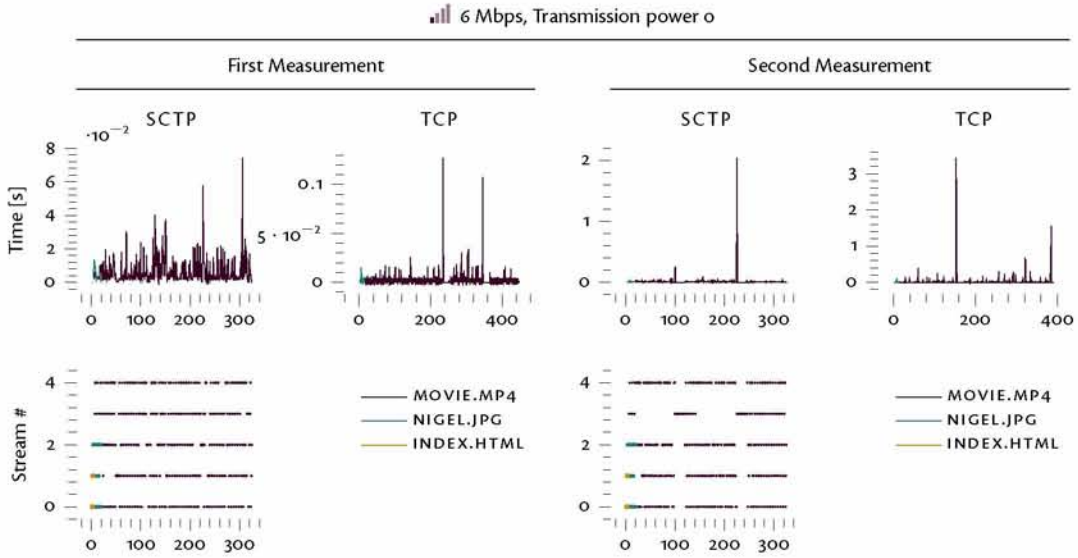


Figure 5.1: Wi-Fi 802.11g

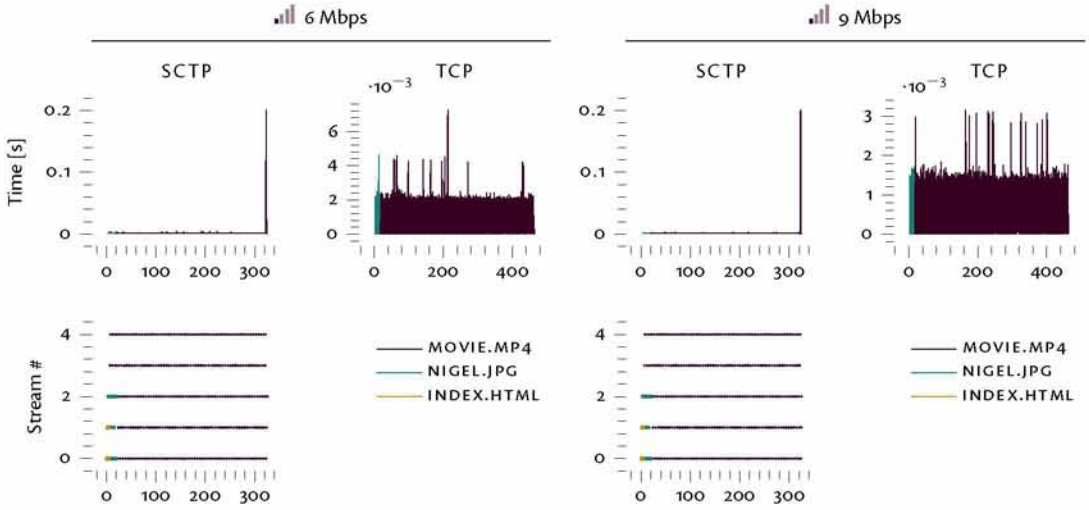


Figure 5.2: Wi-Fi 802.11a

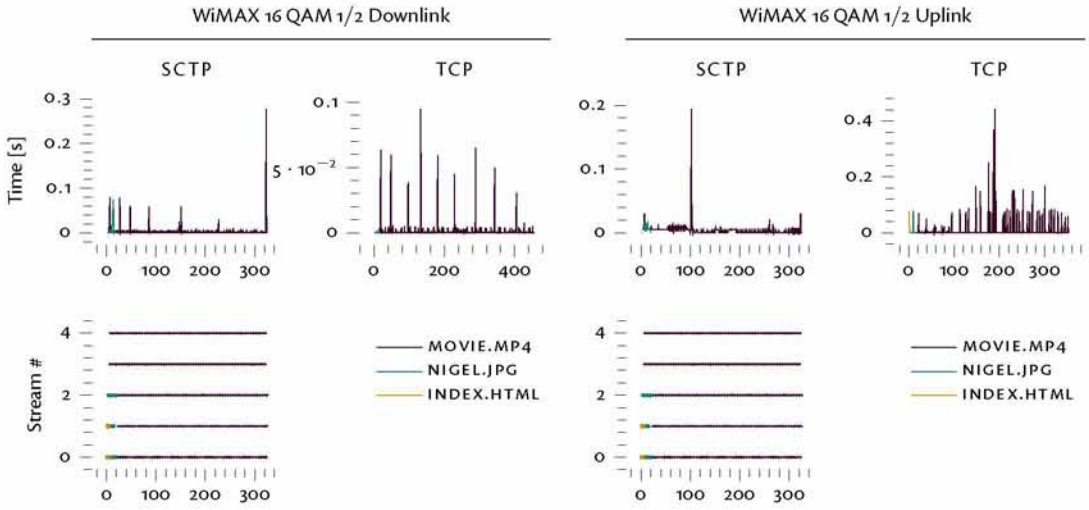


Figure 5.3: WiMAX

6 Conclusion

In this thesis we presented a comparative study of SCTP with TCP over two (2) different technologies: Wi-Fi and WiMAX. The experimentation covered two (2) modes of Wi-Fi (802.11g and 802.11a) and two (2) different topologies on WiMAX (downlink and uplink) with a total of over 550 measurements under real world environment settings. In general, SCTP showed better performance in most of the occasions, with steadier and faster results. There are, though, some specific cases and observations worth mentioning.

As noticed on chapter 5, both protocols had similar behavior and performance on the 802.11a Wi-Fi mode, with slightly better results for the TCP. This is a pretty much expected behavior, since 802.11a is interference free and the network state is as close as possible to ideal. That means that the TCP protocol does not suffer from the Head of Line Blocking effect, as packet losses are scarce. In combination with the fact that TCP is a much older and widely used protocol and therefore significantly improved over the years, it performs great against SCTP.

In the majority of the measurement plots for SCTP over WiMAX, repeating spikes can be spotted throughout the diagrams. Fact is that the software buffer of UNIX for the WIMAX traffic is drained and thus creates delays while sending data over the WIMAX link, which causes that pattern on all measurements.

Another noticeable SCTP behavior shown on the diagrams is a huge spike towards the end of the measurement that can be seen on almost every SCTP diagram, regardless the experiment specifications. More specifically, this spike corresponds to chunk number 323 of the video file ("movie.mp4"), the second to last chunk sent on each experiment. Referring to section 4.6 about the SCTP implementation, there is a mechanism for reconstructing the out-of-order chunks received during the experiment. That procedure takes place at that exact moment, where all out-of-order segments are re-ordered and written into the right place into the corresponding binary file, which explains the delay depicted on the plots.

Concluding, some general notices, deriving once again from the diagrams. After thoroughly examining all the plots and results, TCP seems to perform better when using high transmission power at low transmission rates. Moving to higher rates, TCP starts to suffer from serious Head of Line Blocking, and its performance starts to fall behind compared to SCTP. On the other hand, SCTP deals more effectively with often packet losses, taking advantage of its multi-stream feature and limiting the Head of Line Blocking effect to specific streams.

7 Bibliography

Budigere, K. (2013). Linux Implementation Study of Stream Control Transmission Protocol, in *Proceedings of Seminar on Network Protocols in Operating Systems*, 22.

Cano, M.-D. (2011). On the Use of sctp in Wireless Networks, *Recent Advances in Wireless Communications and Networks*, 12, 247–266.

Griffith, N. (2006). nweb: a tiny, safe Web server (static pages only), http://www.ibm.com/developerworks/systems/library/es-nweb/.

Heinz, I. and J. Gerard (2003). Priorities in stream transmission control protocol (sctp) multistreaming, Tech. rep., DTIC Document.

Hurtig, P. (2008). Improving the Timeliness of sctp Message Transfers, Licentiate thesis, Department of Computer Science, Karlstad University.

Natarajan, P., J. R. Iyengar, P. D. Amer, and R. Stewart (2006). sctp: an innovative transport layer protocol for the web, in *Proceedings of the 15th international conference on World Wide Web*, ACM, 615–624.

Nelson, V., J. Tania, and H. Yezekael (2008). Performance of sctp in Wi-Fi and WiMAX networks with multi-homed mobiles, in *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 71.

Ravier, T., R. Brennan, and T. Curran (2001). Experimental studies of sctp multi-homing, *Teltec DCU, Dublin*, 9.

Stewart, R. and C. Metz (2001). sctp: new transport protocol for tcp/IP, *Internet Computing, IEEE*, 5(6), 64–69.

Tufte, E. R. (2001). *The Visual Display of Quantitative Information*, Cheshire, Connecticut: Graphics Press.

———— (2006). *Beautiful Evidence*, Cheshire, Connecticut: Graphics Press, LLC, first ed.

University of Thessaly (—). Network Implementation Testbed Laboratory of the Computer and Communication Engineering Department at University of Thessaly, http://nitlab.inf.uth.gr/NITlab/.

Wang, Y., I. Rhee, and S. Ha (2011). Augment sctp multi-streaming with pluggable scheduling, in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, IEEE, 810–815.

Ye, G., T. Saadawi, and M. Lee (2002). sctp congestion control performance in wireless multi-hop networks, in *MILCOM 2002. Proceedings*, IEEE, vol. 2, 934–939.

A Appendix

A.1 Wi-Fi 802.11g

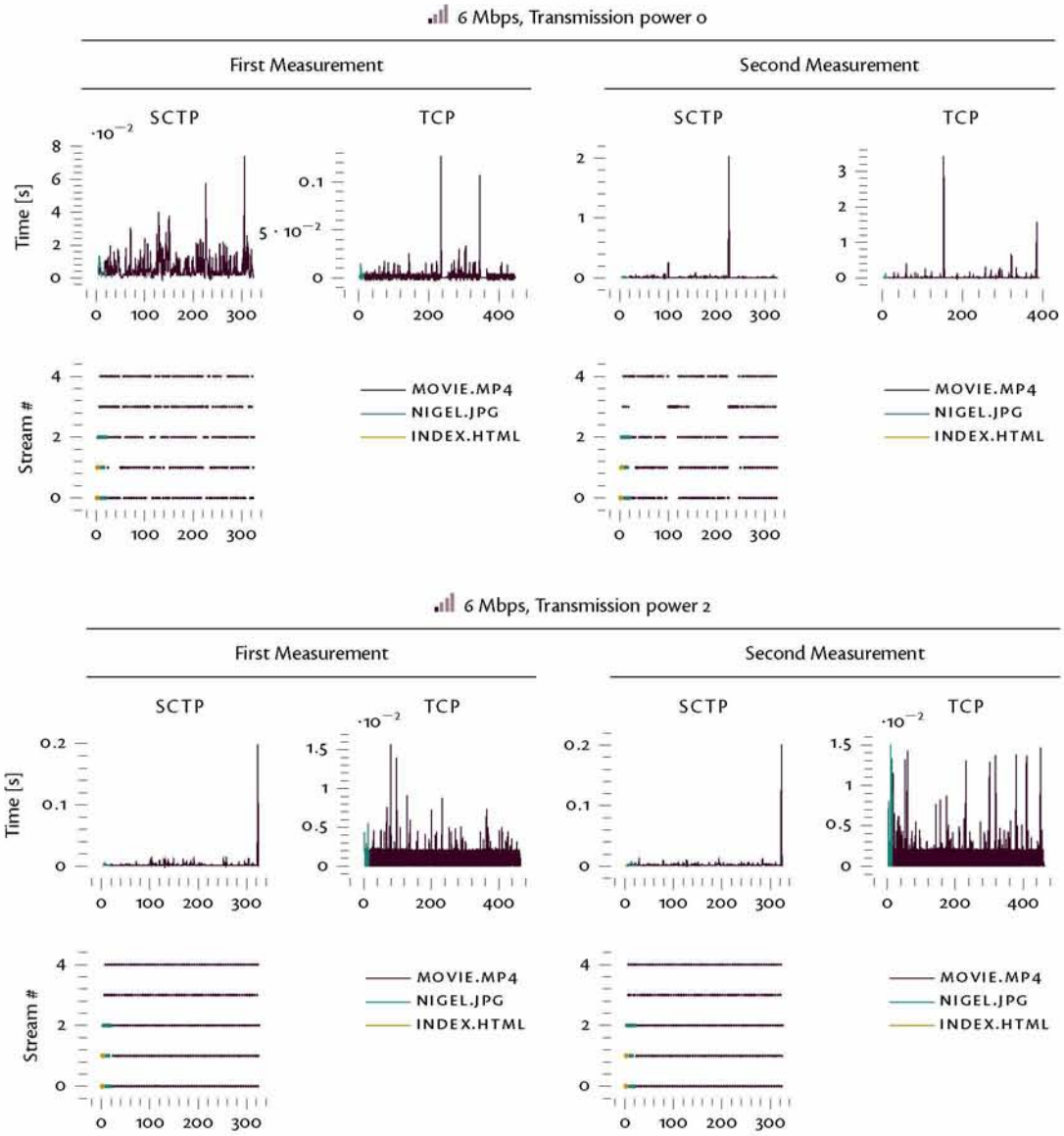


Figure A.1: Experimental measurements plots for the Wi-Fi 802.11g protocol (part I)

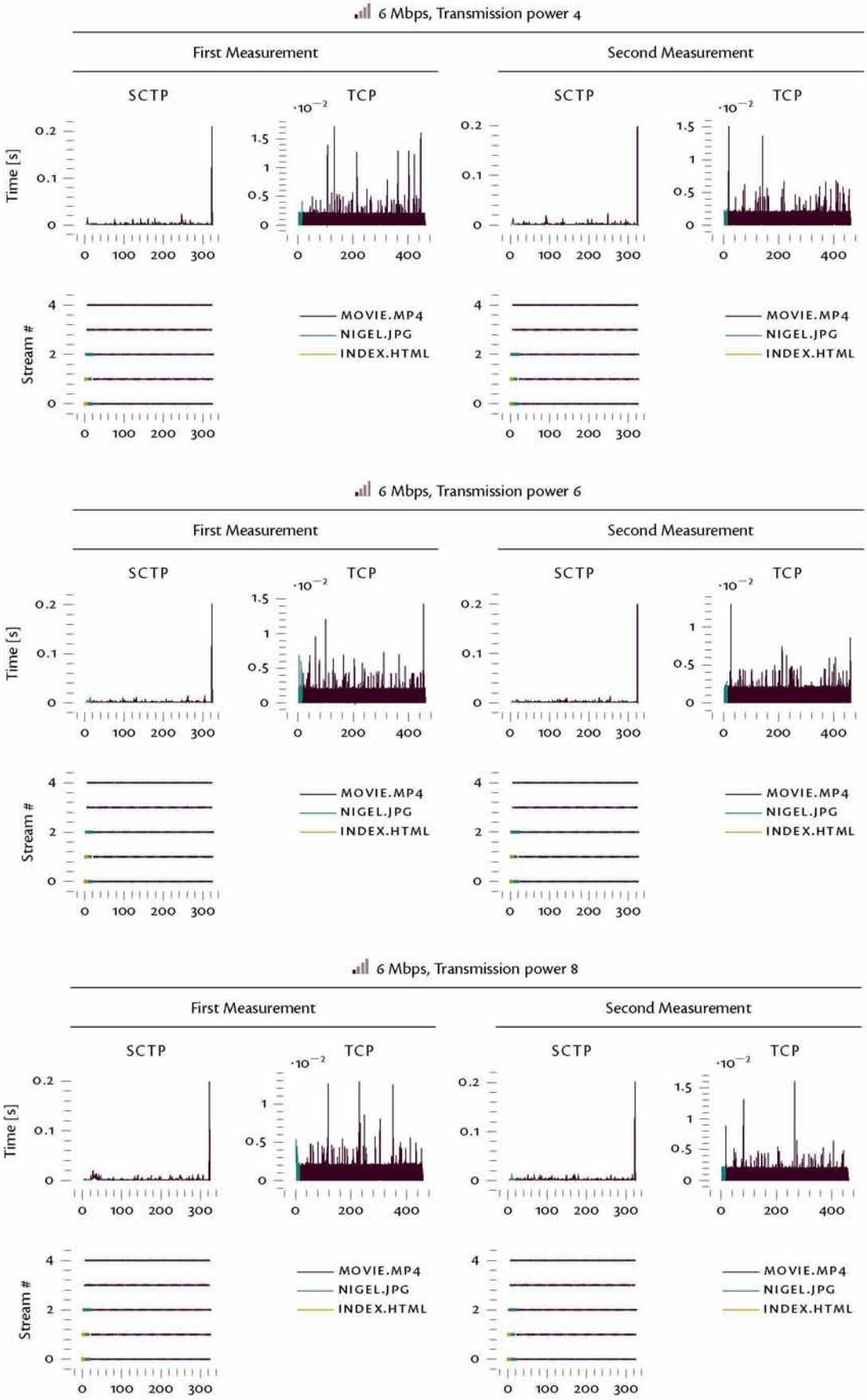


Figure A.2: Experimental measurements plots for the Wi-Fi 802.11g protocol (part II)

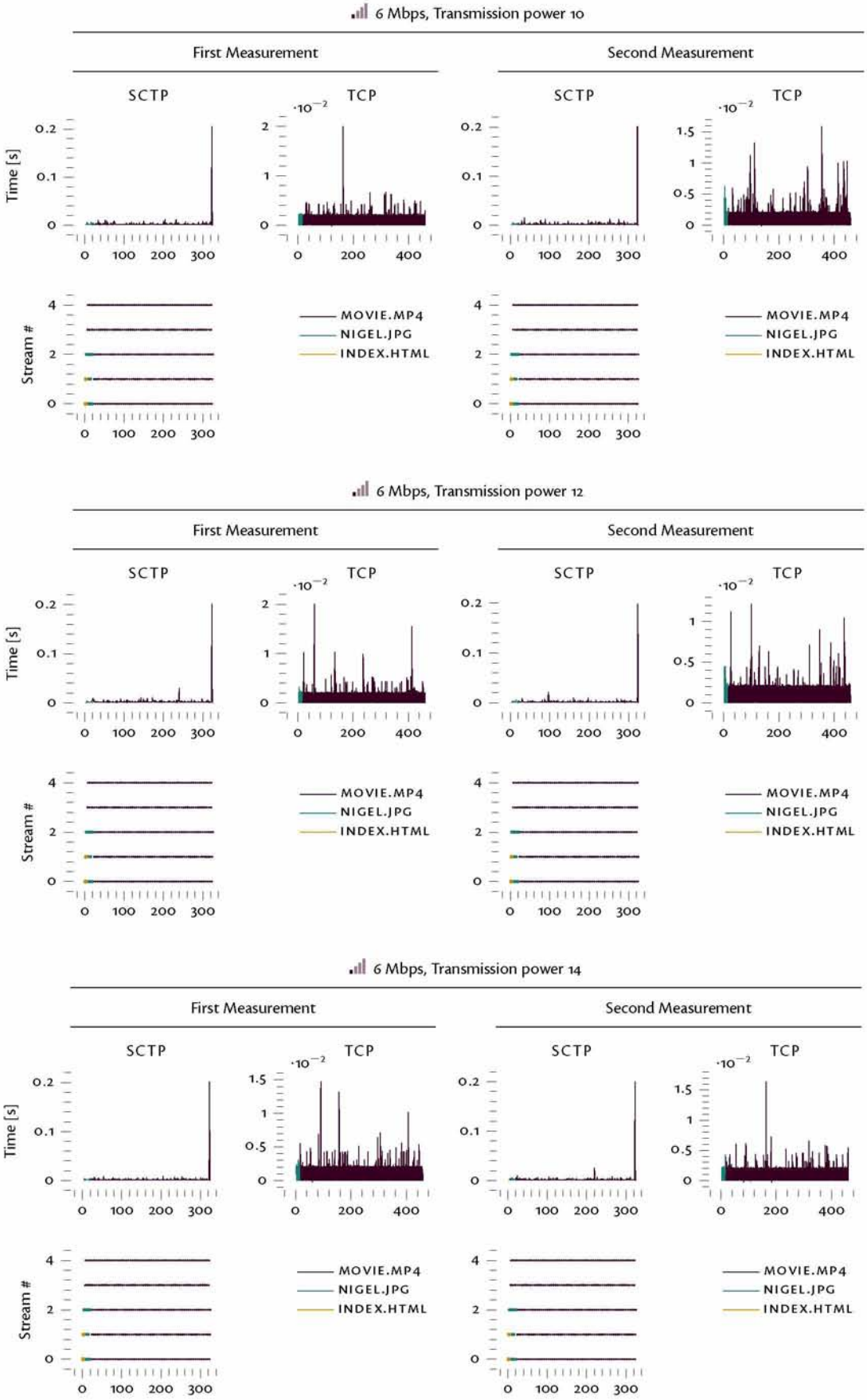


Figure A.3: Experimental measurements plots for the Wi-Fi 802.11g protocol (part III)

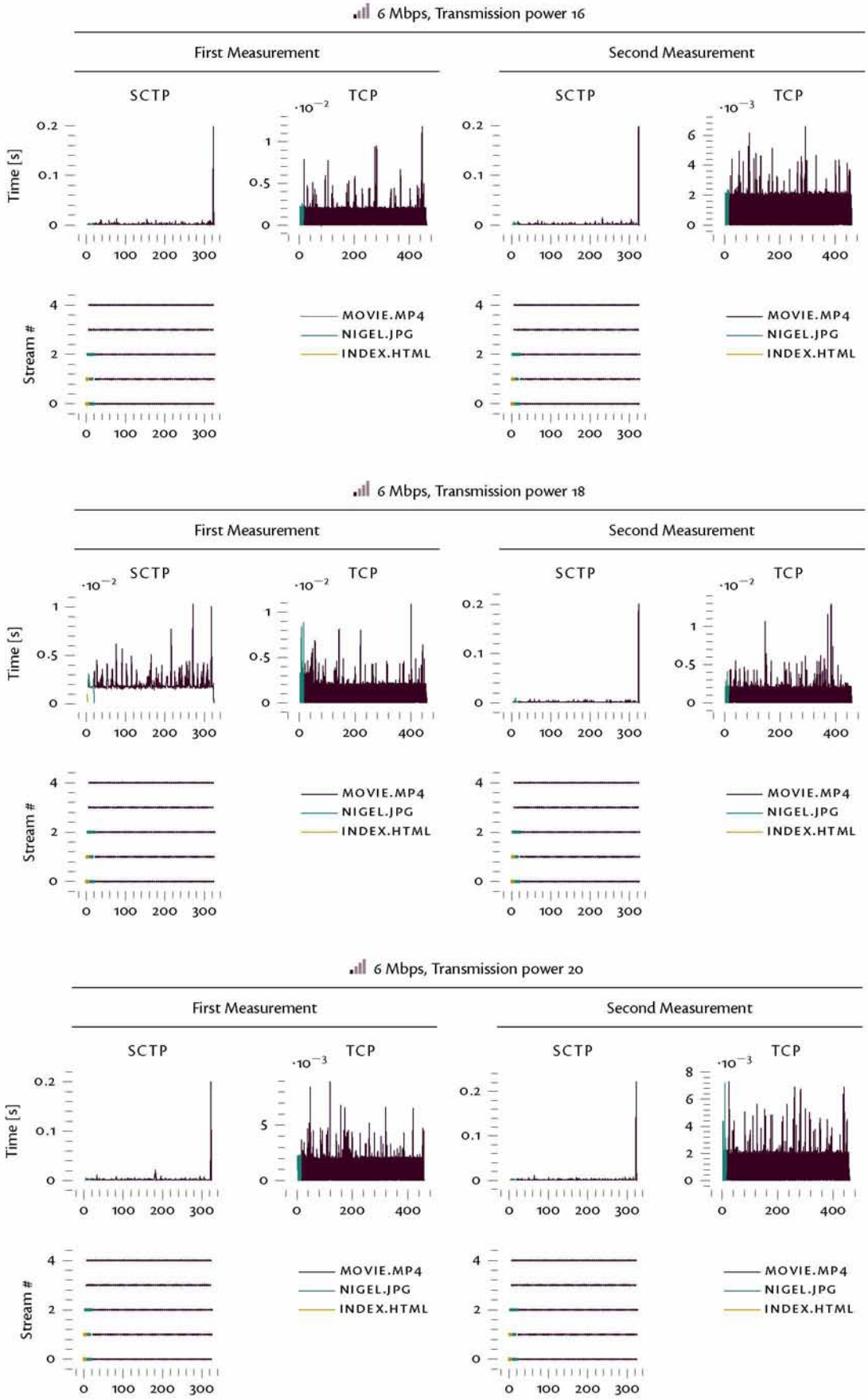


Figure A.4: Experimental measurements plots for the Wi-Fi 802.11g protocol (part IV)

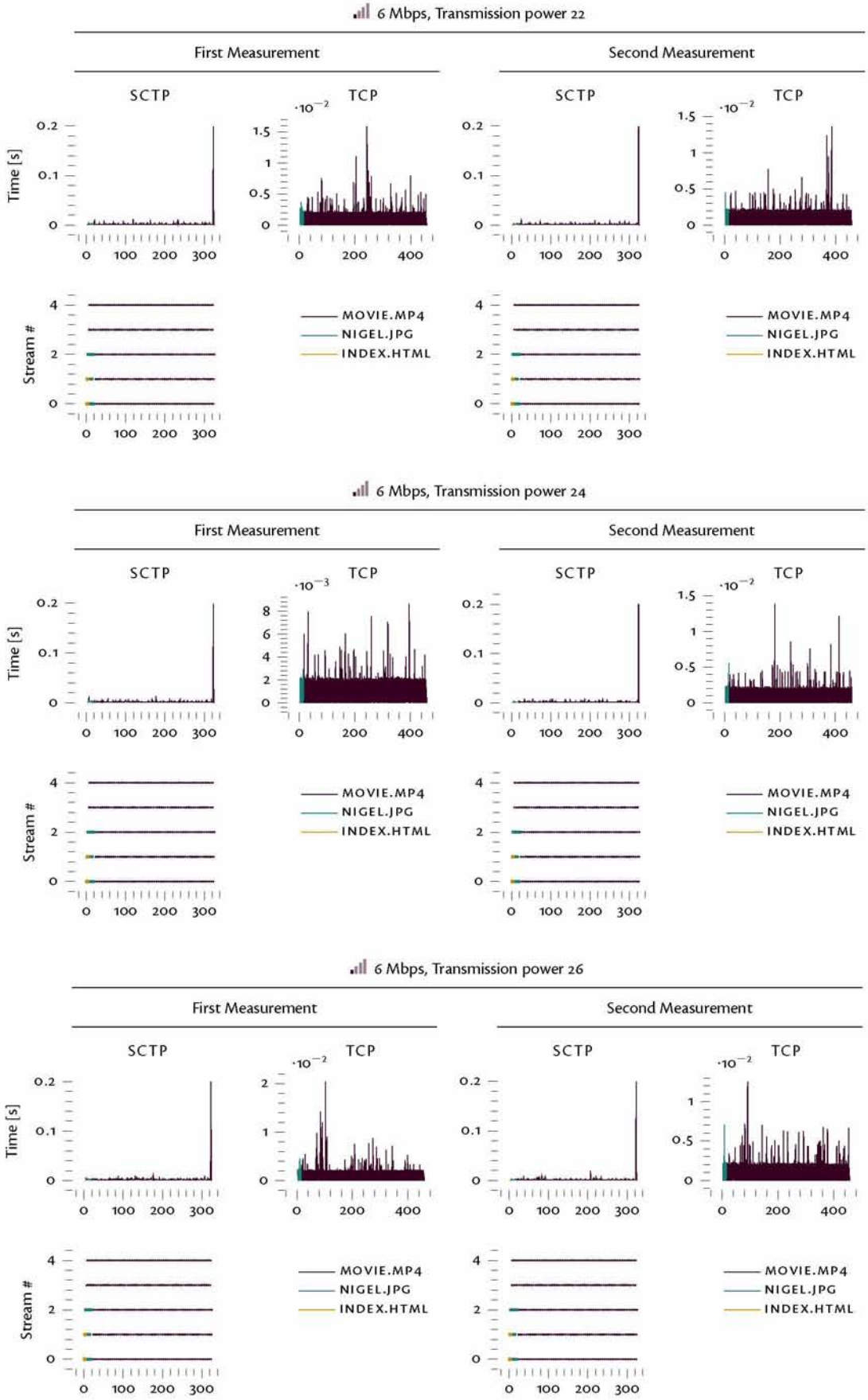


Figure A.5: Experimental measurements plots for the Wi-Fi 802.11g protocol (part V)

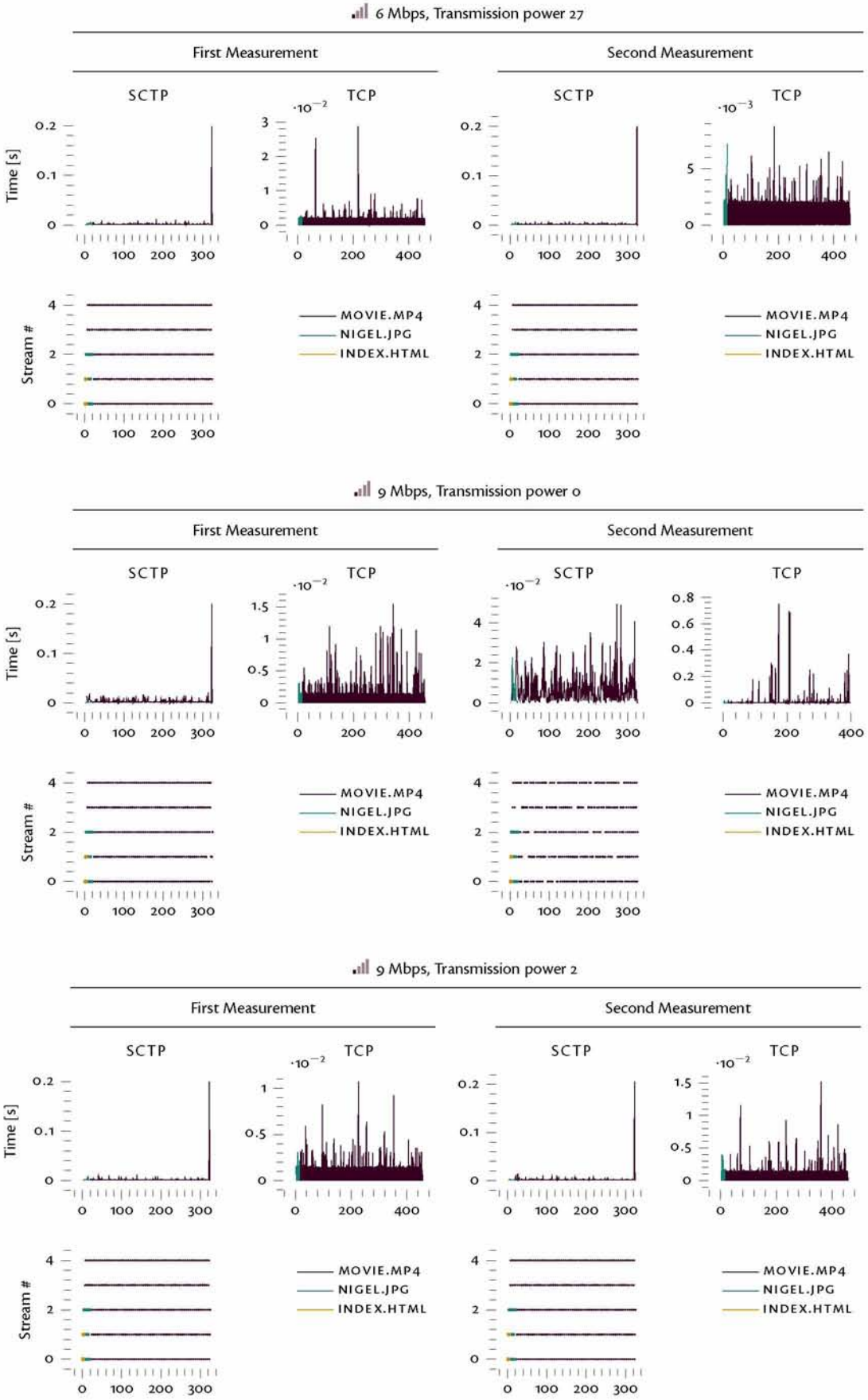


Figure A.6: Experimental measurements plots for the Wi-Fi 802.11g protocol (part VI)

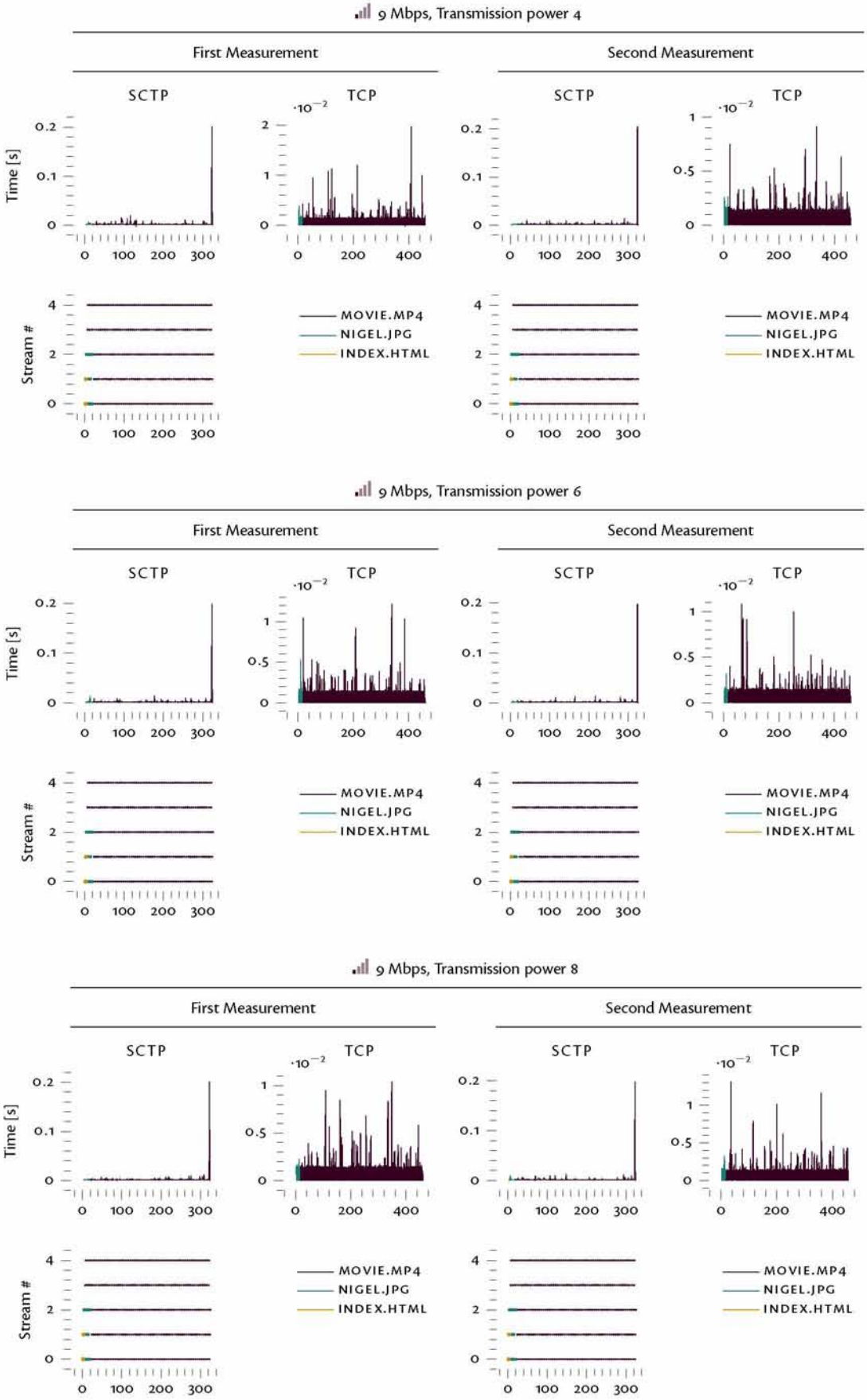


Figure A.7: Experimental measurements plots for the Wi-Fi 802.11g protocol (part VII)

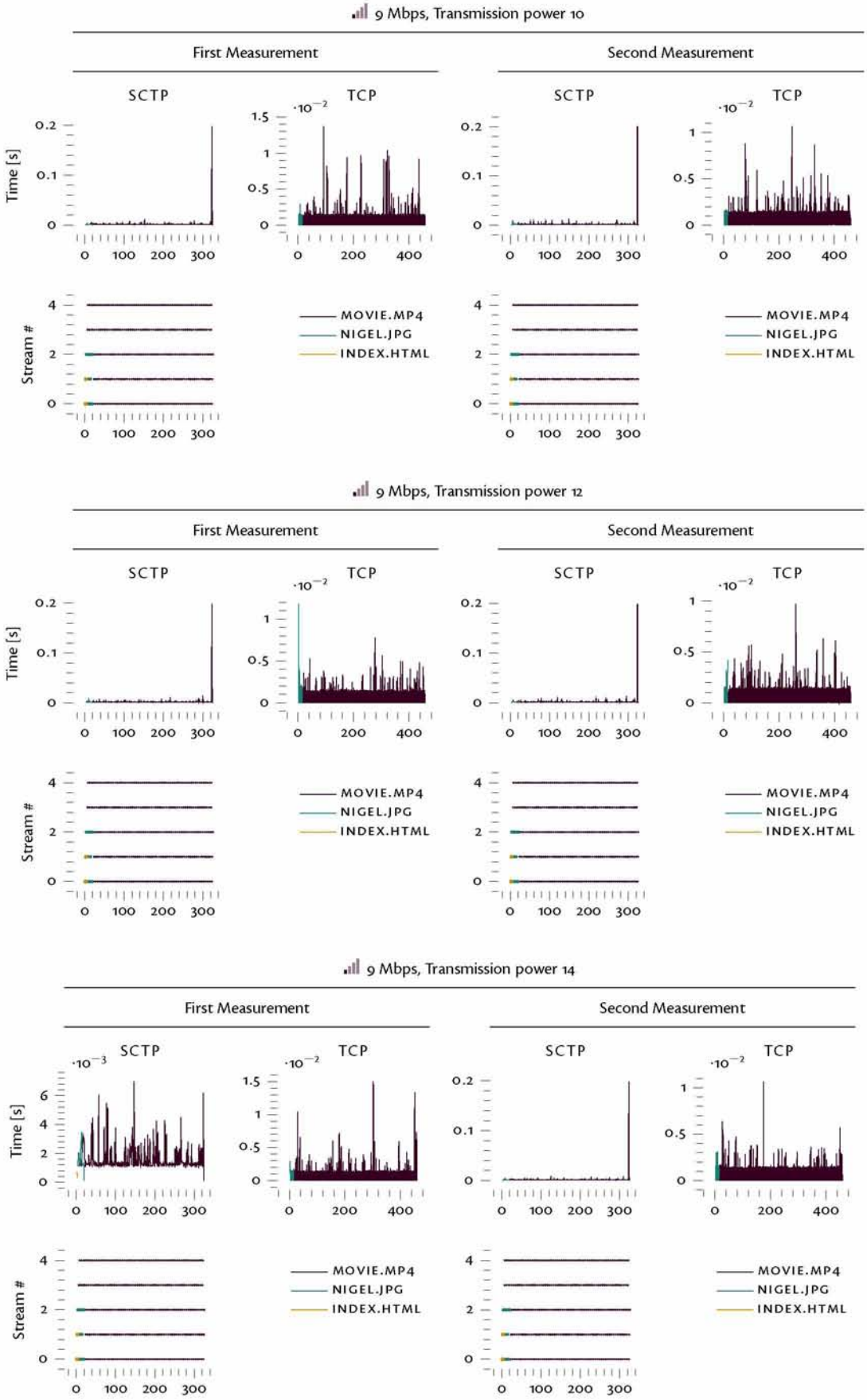


Figure A.8: Experimental measurements plots for the Wi-Fi 802.11g protocol (part VIII)

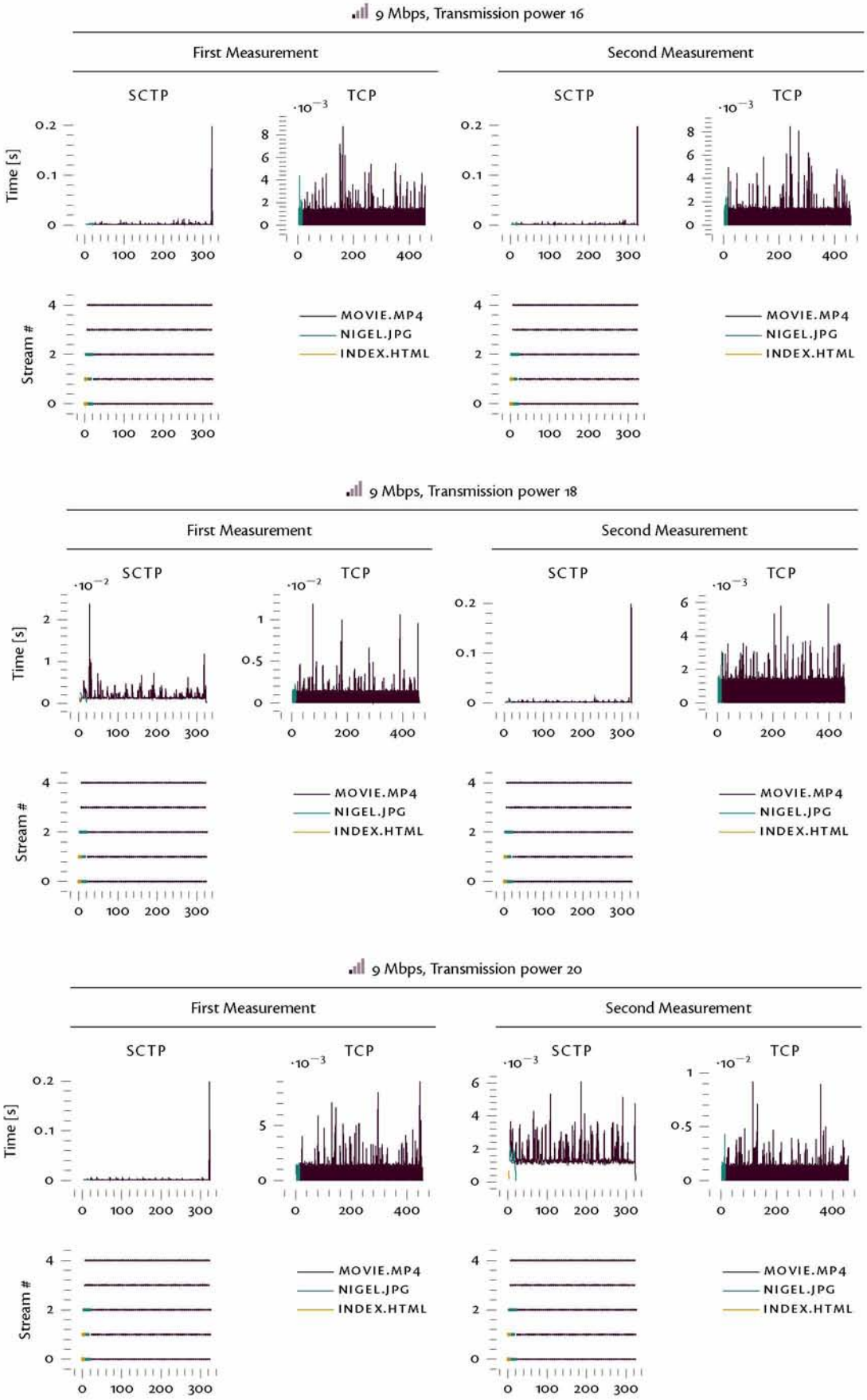


Figure A.9: Experimental measurements plots for the Wi-Fi 802.11g protocol (part IX)

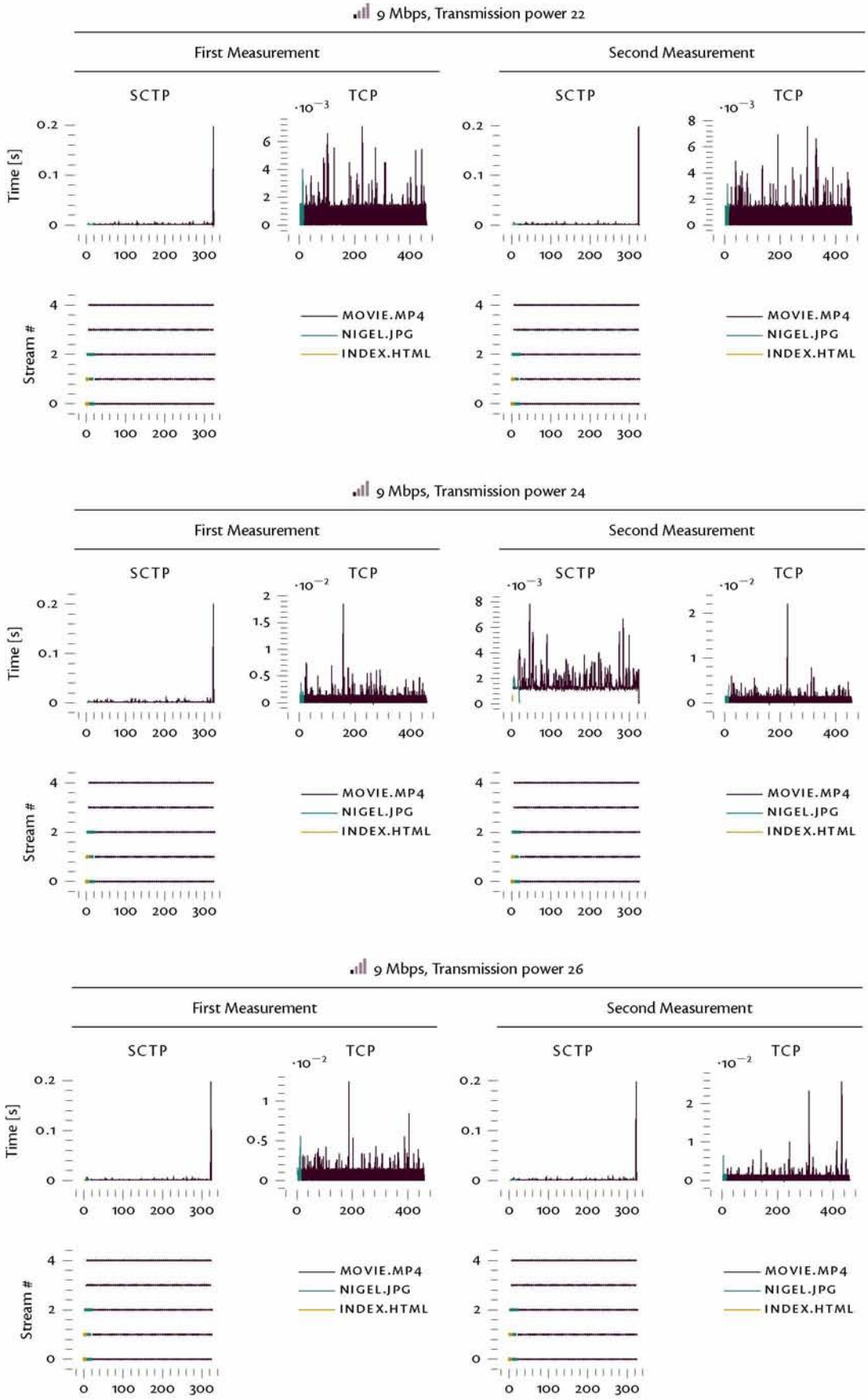


Figure A.10: Experimental measurements plots for the Wi-Fi 802.11g protocol (part X)

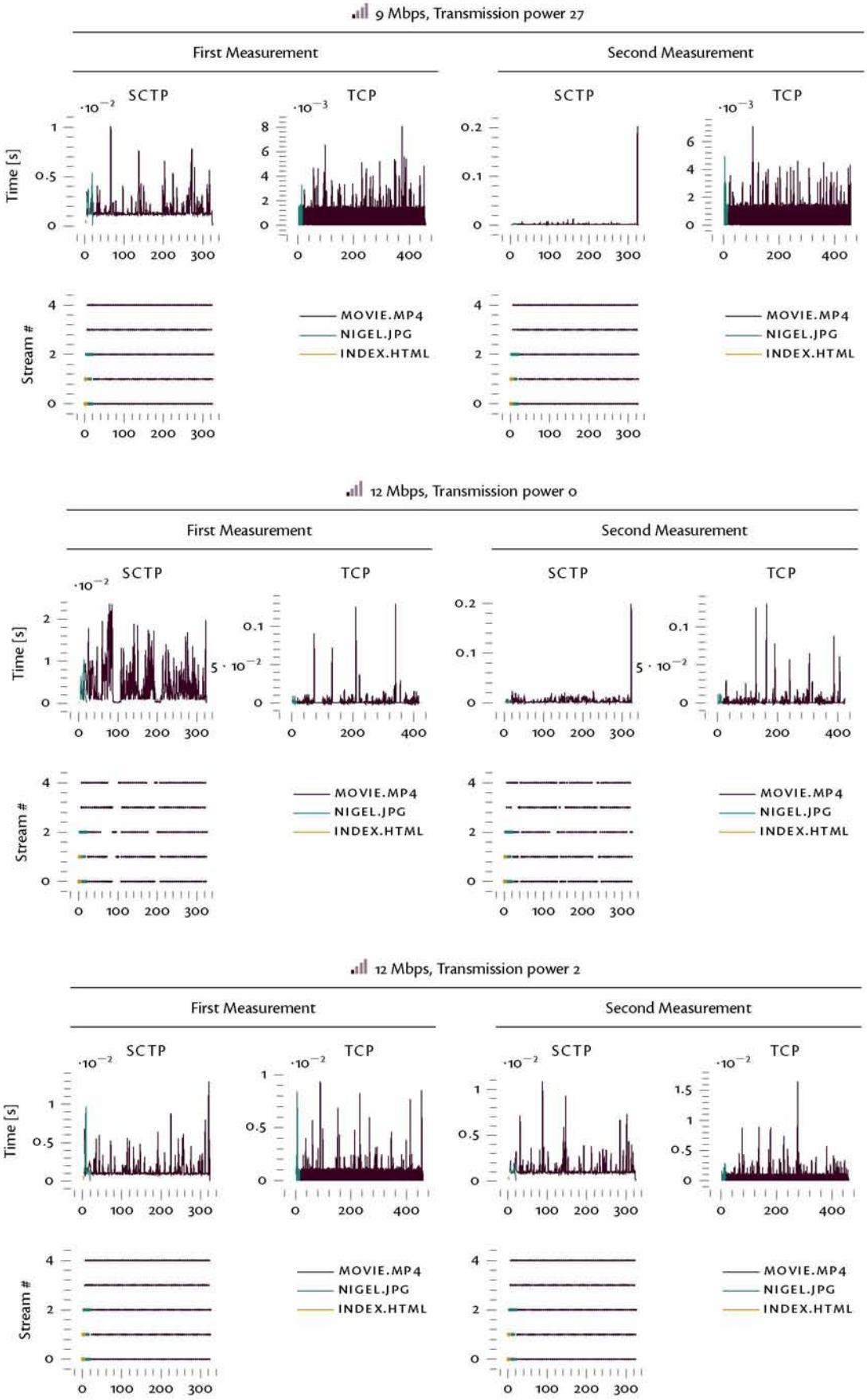


Figure A.11: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XI)

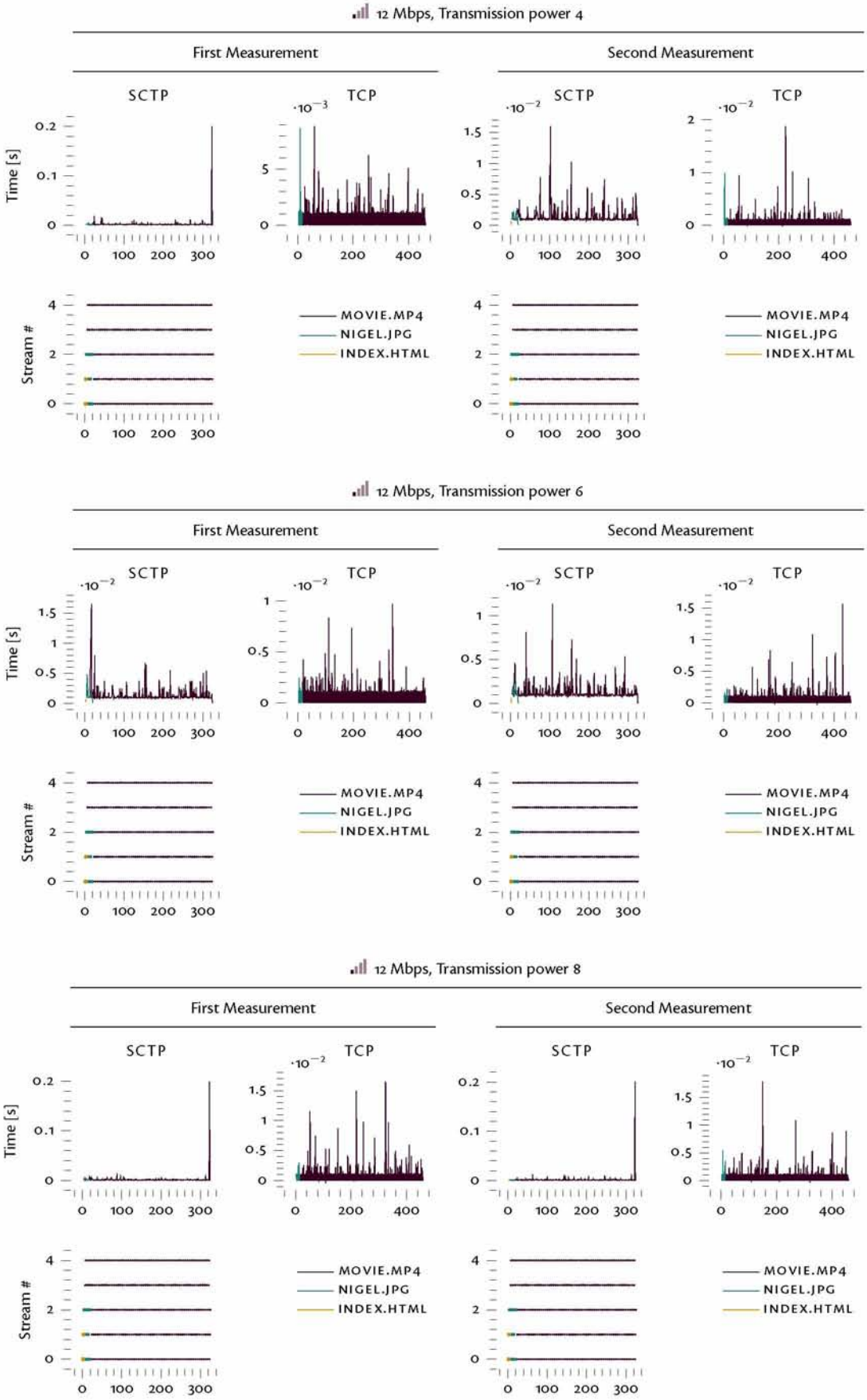


Figure A.12: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XII)

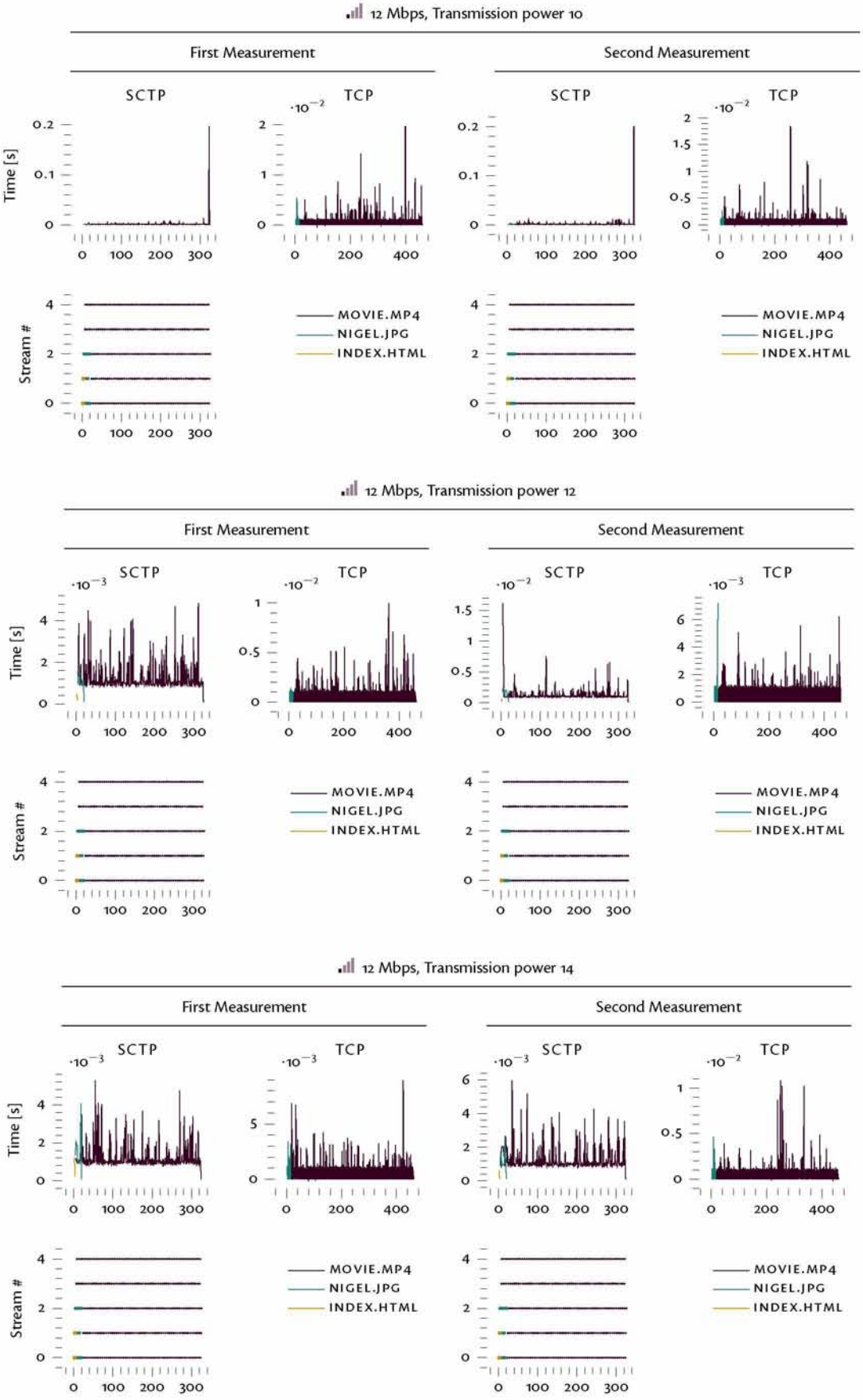


Figure A.13: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XIII)

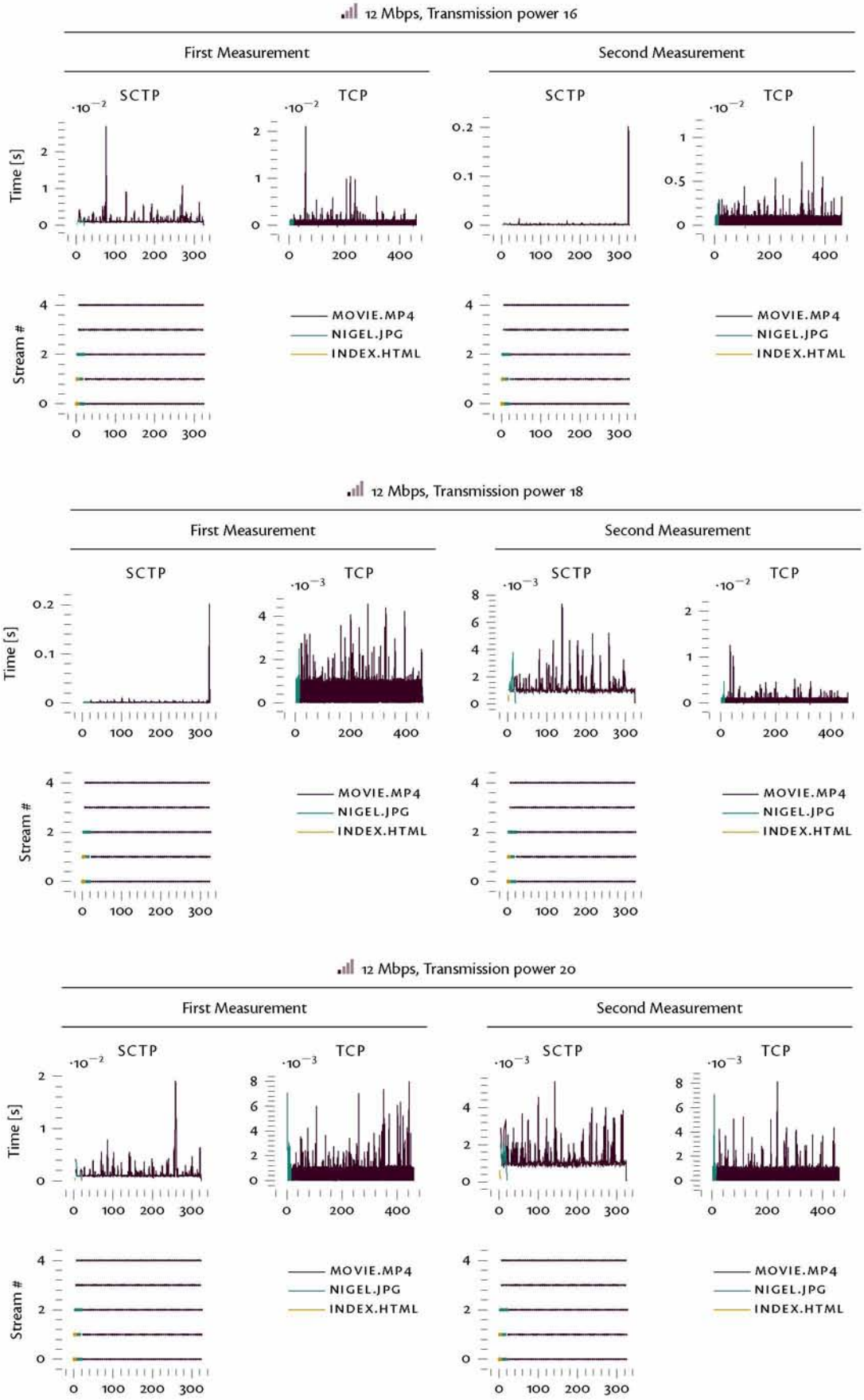


Figure A.14: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XIV)

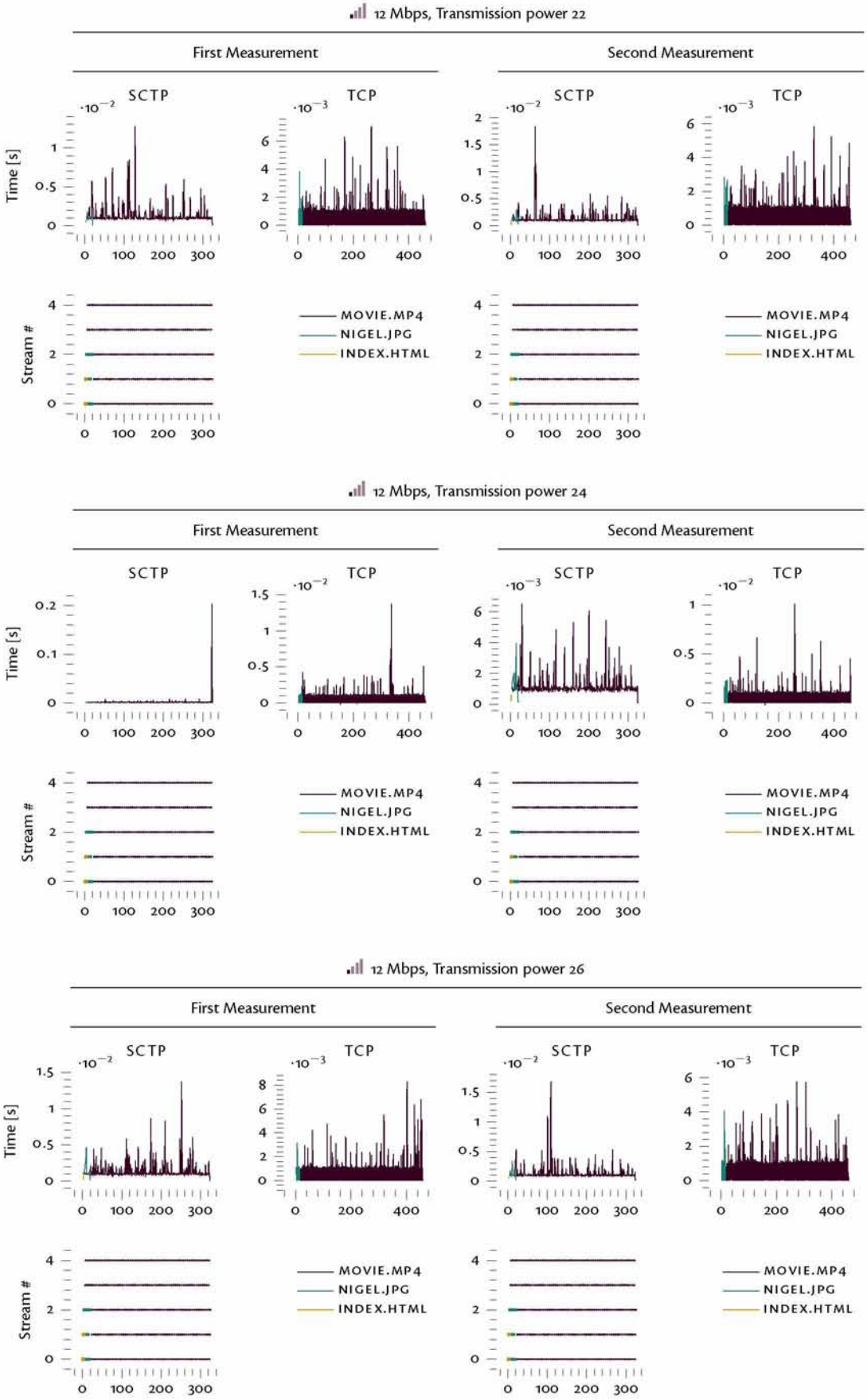


Figure A.15: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XV)

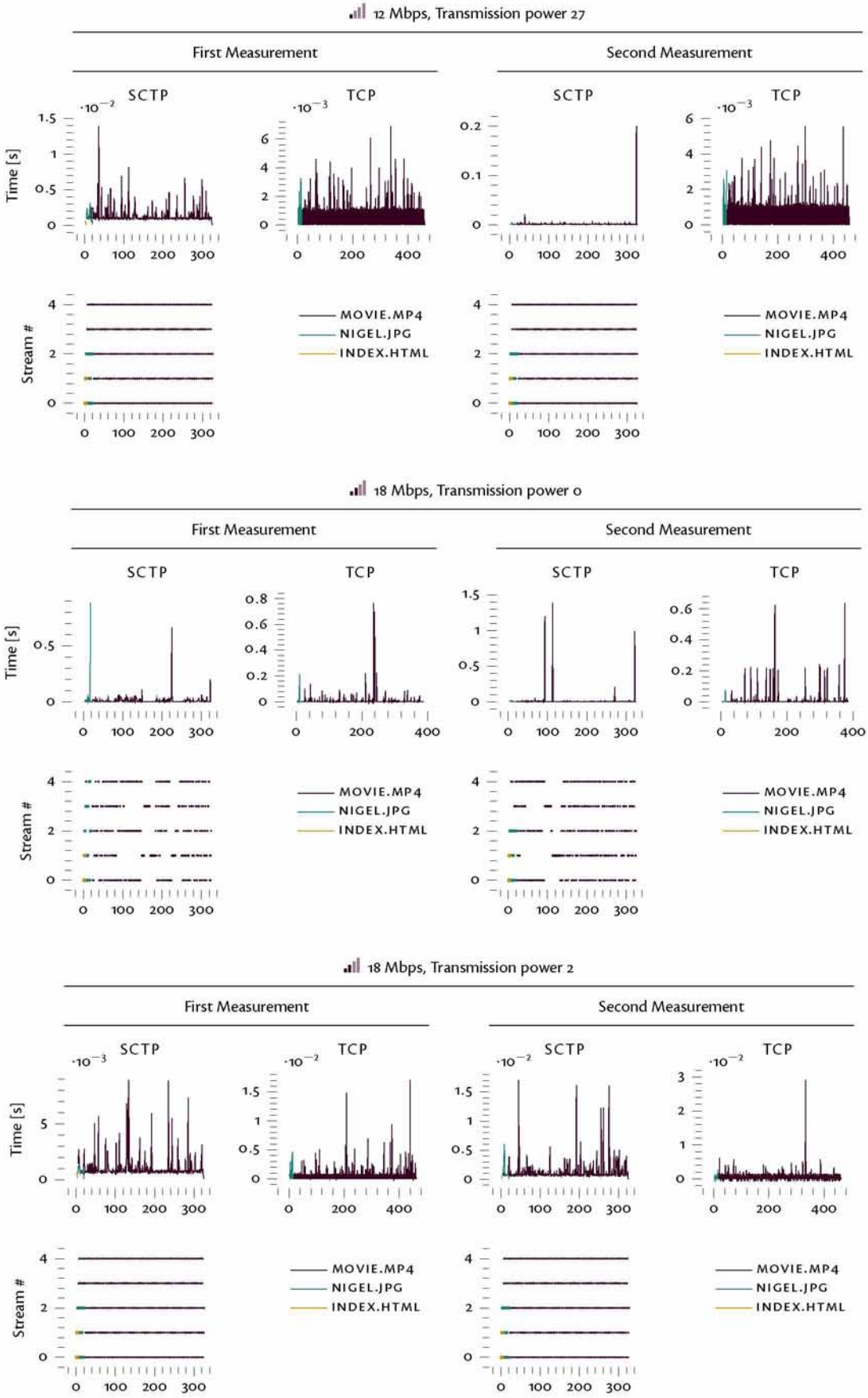


Figure A.16: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XVI)

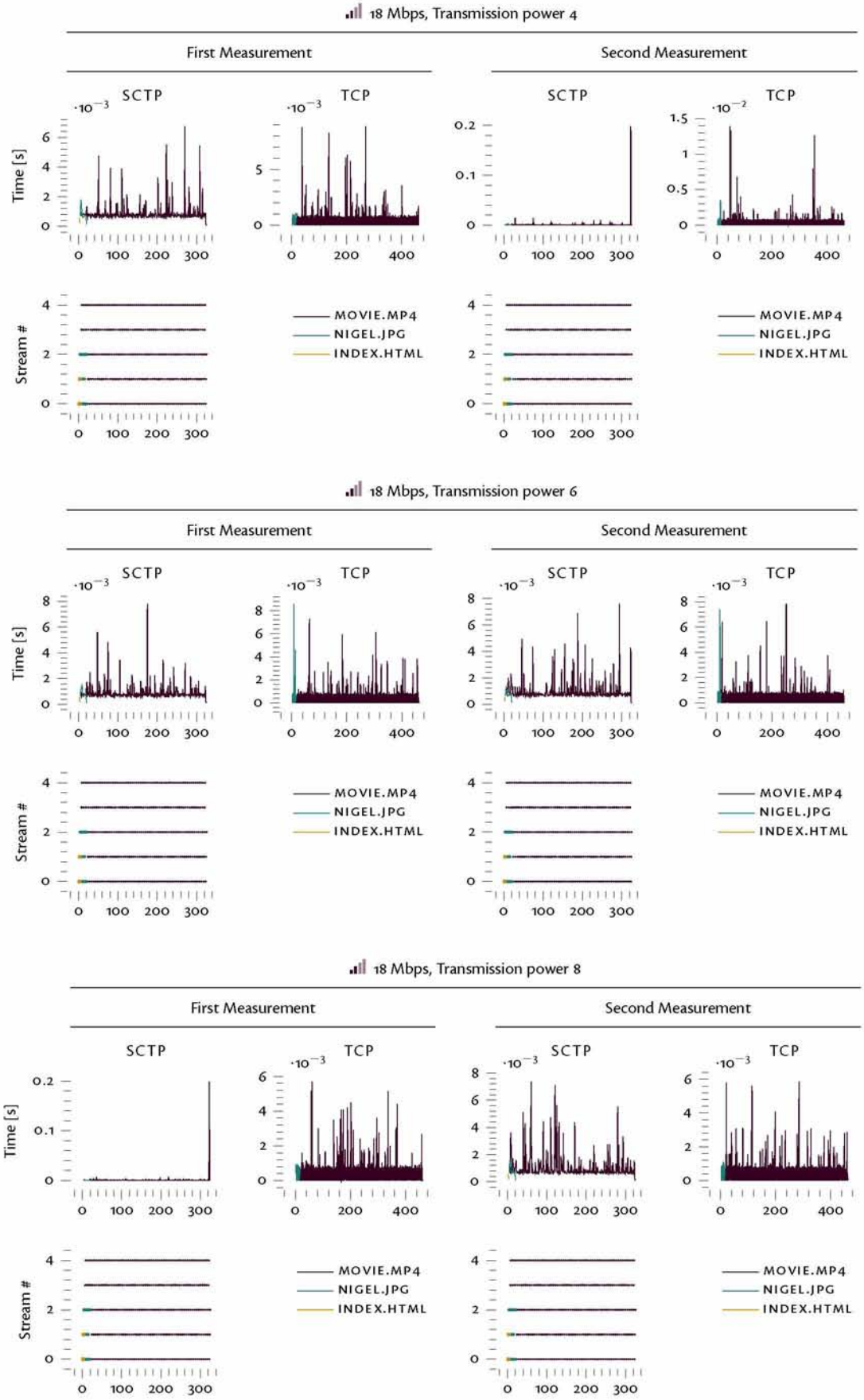


Figure A.17: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XVII)

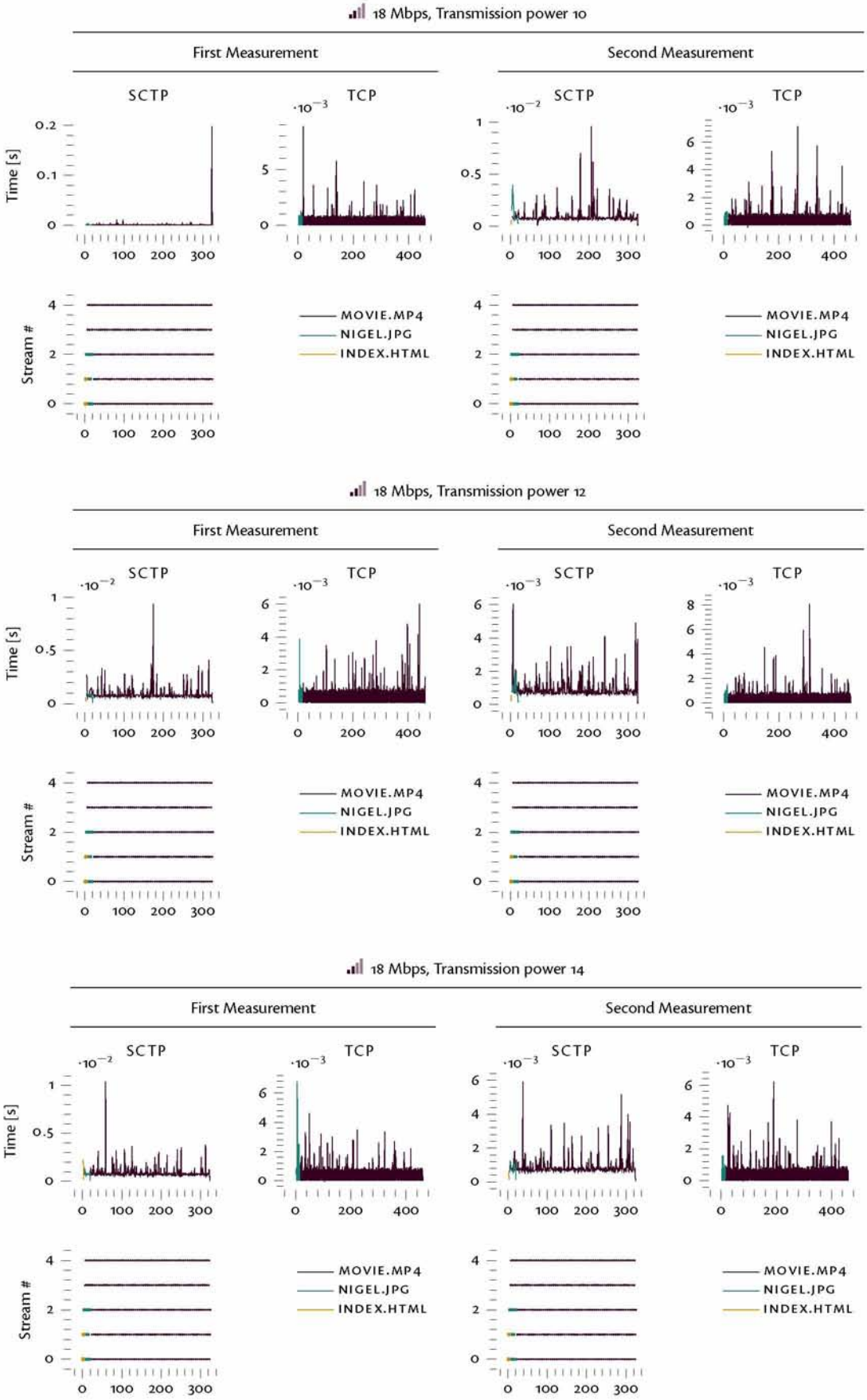


Figure A.18: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XVIII)

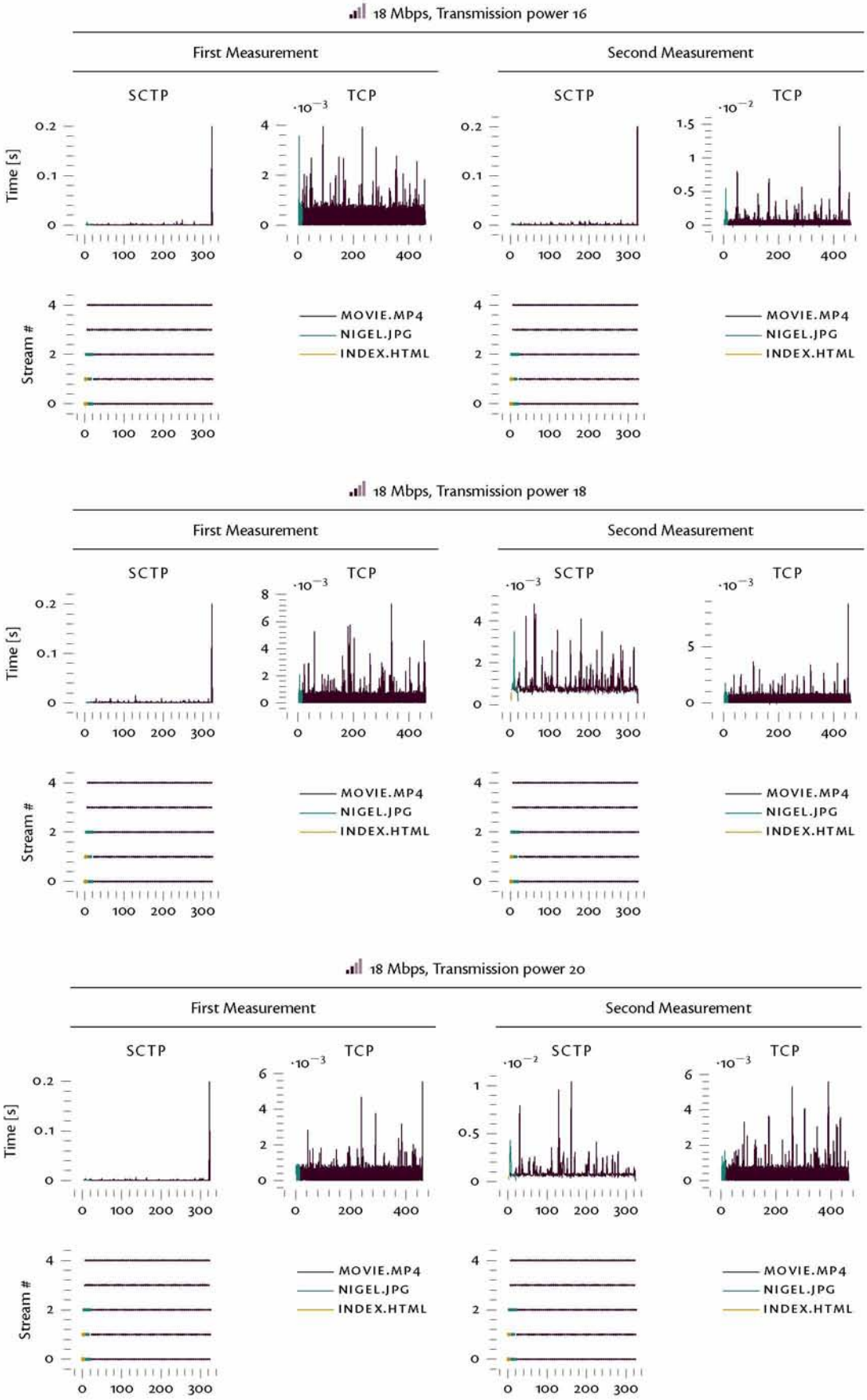


Figure A.19: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XIX)

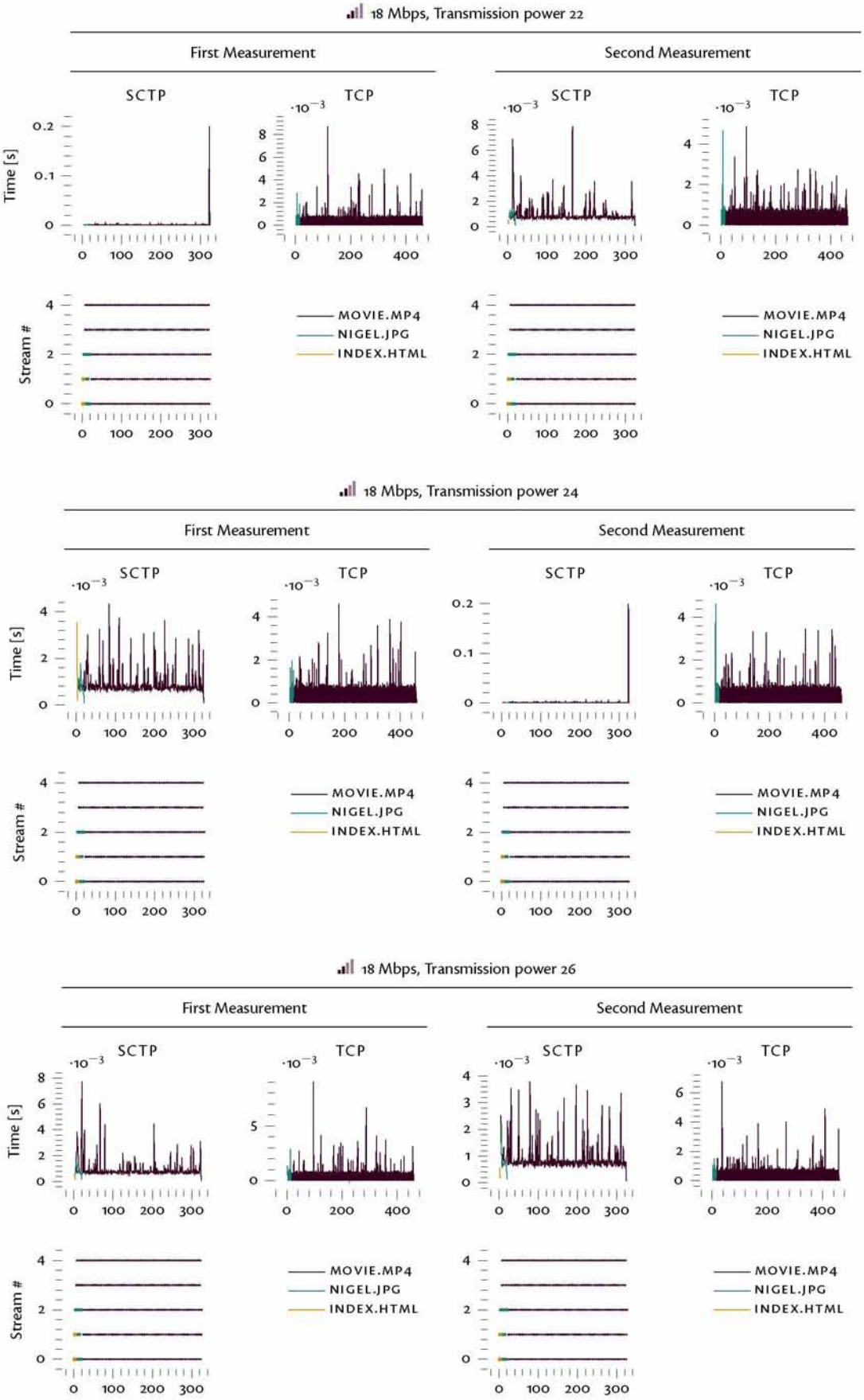


Figure A.20: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XX)

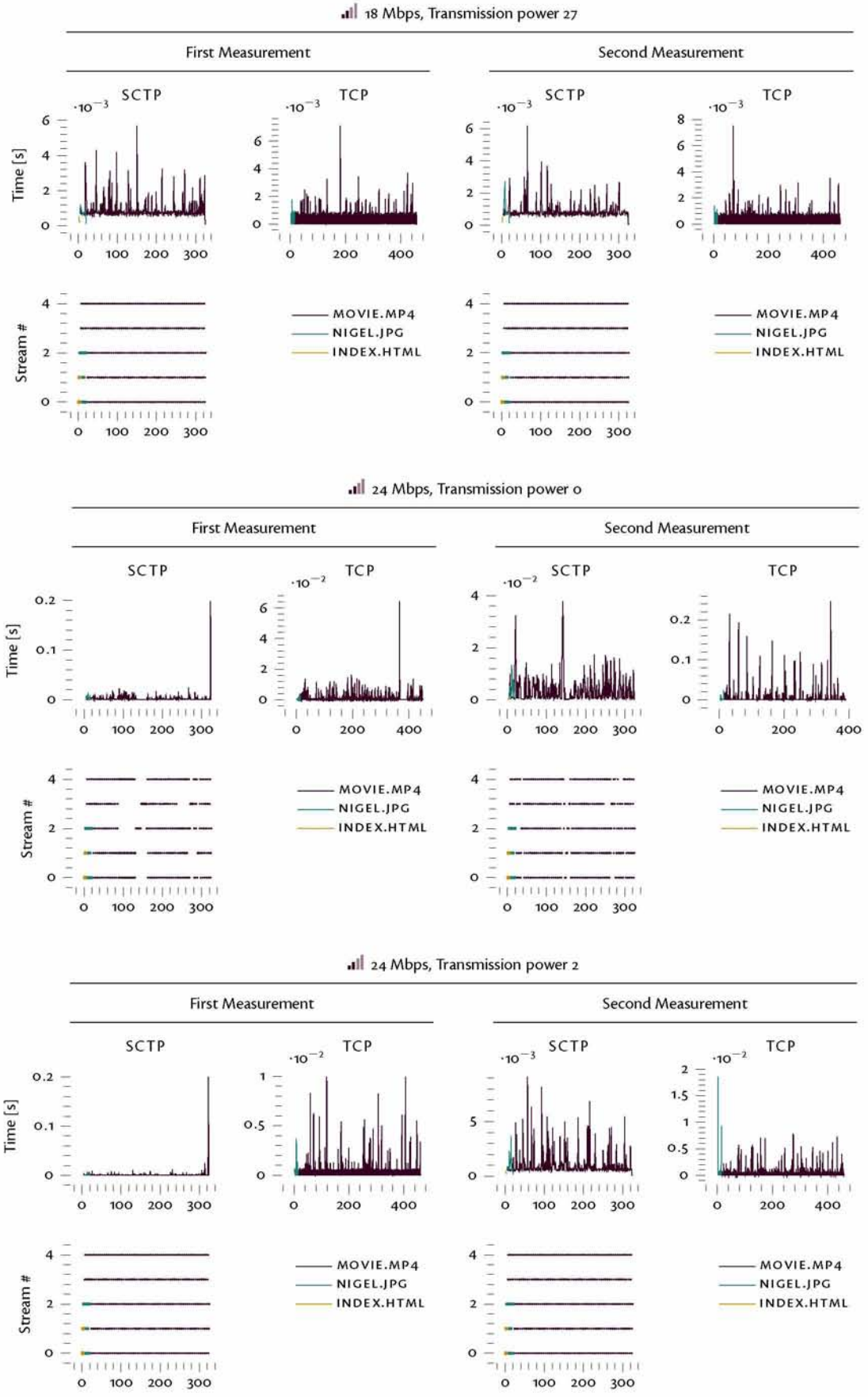


Figure A.21: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXI)

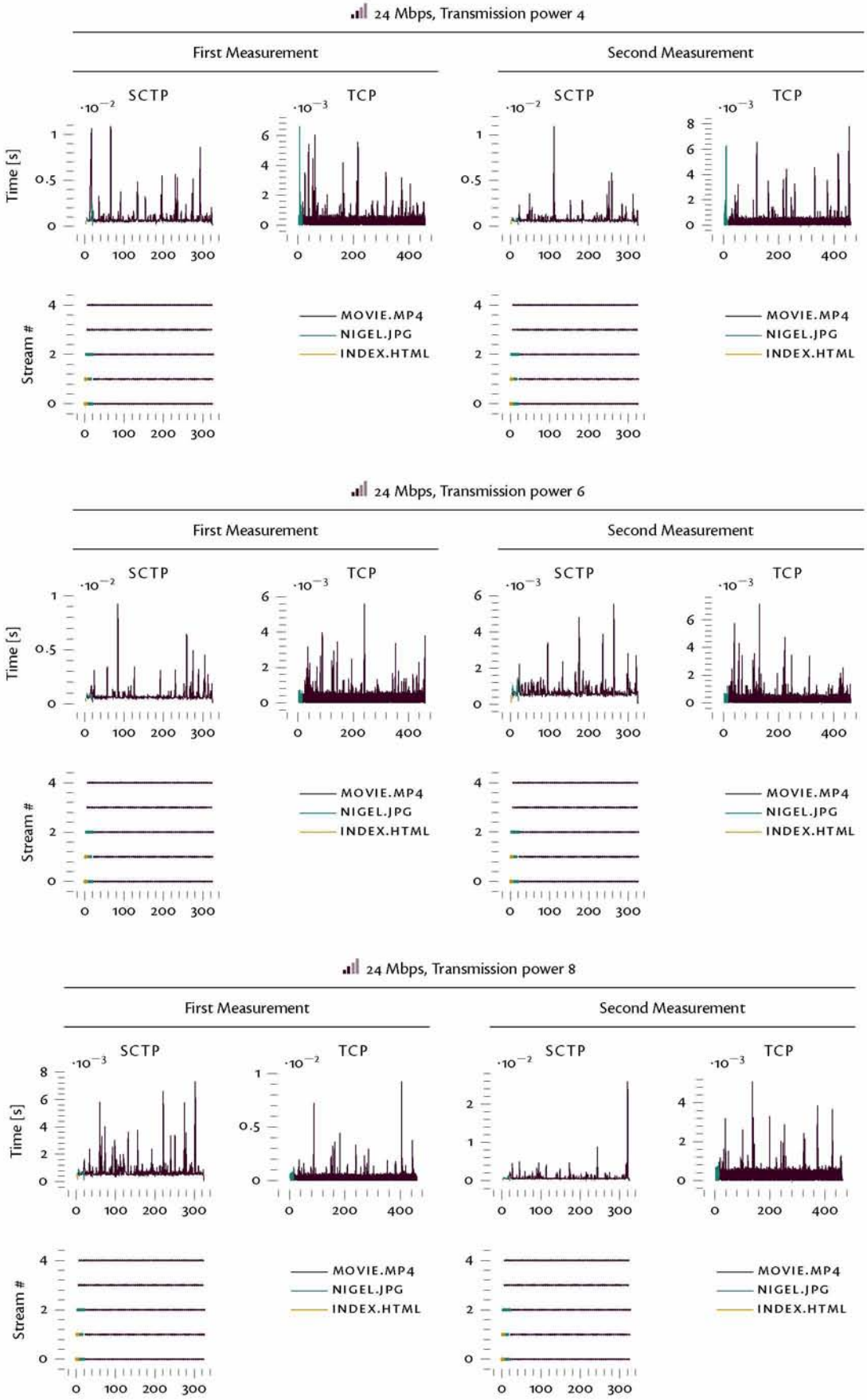


Figure A.22: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXII)

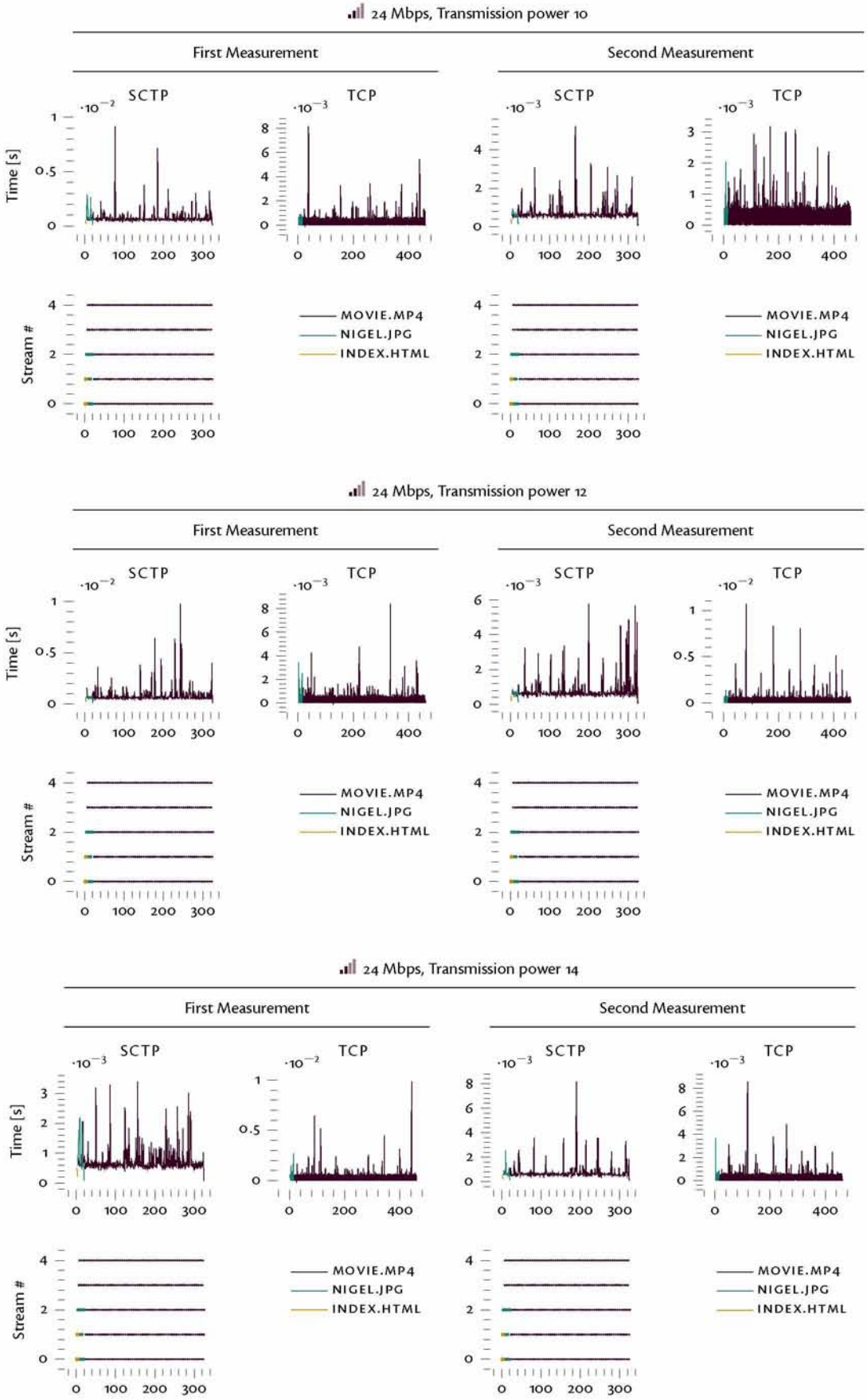


Figure A.23: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXIII)

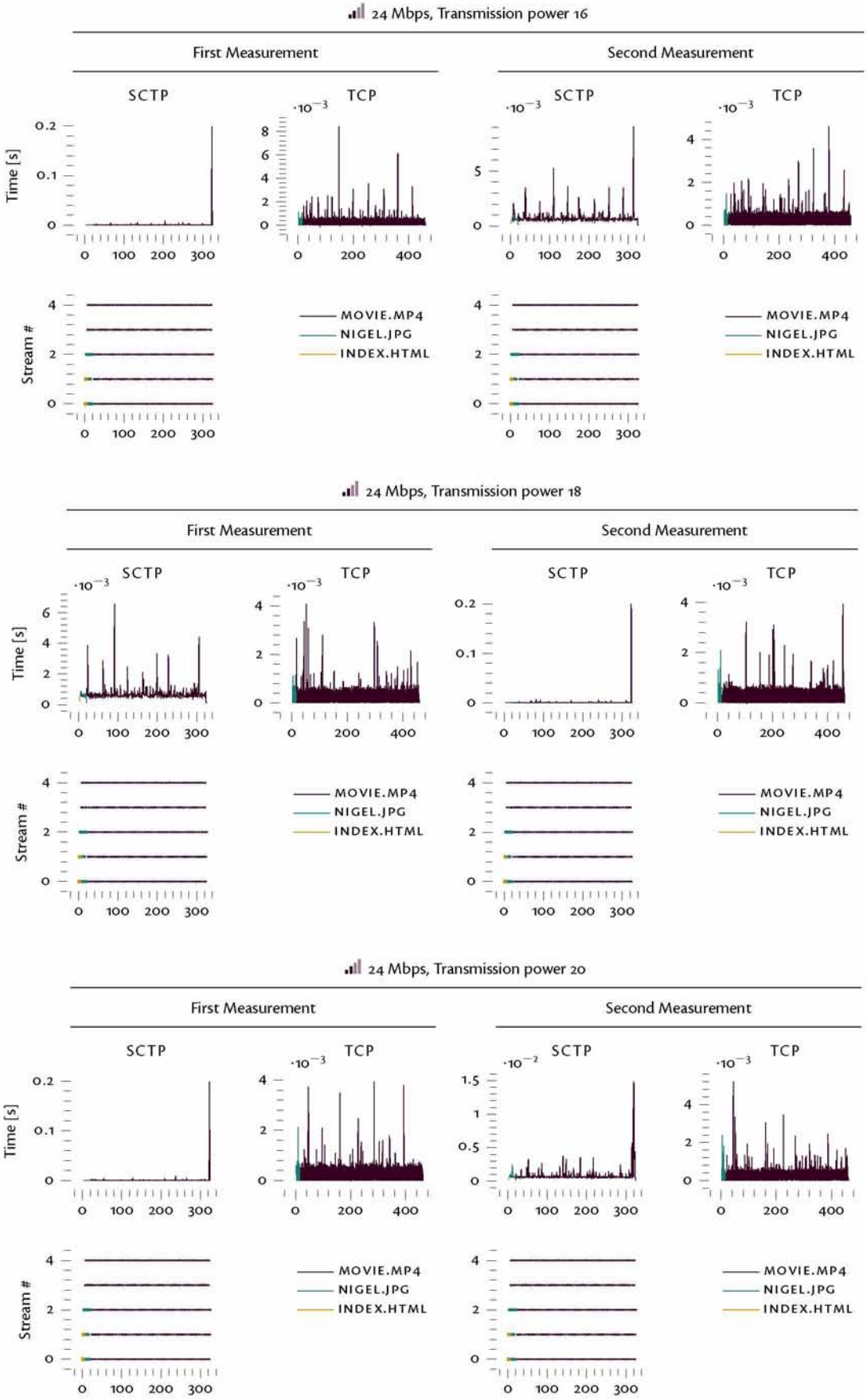


Figure A.24: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXIV)

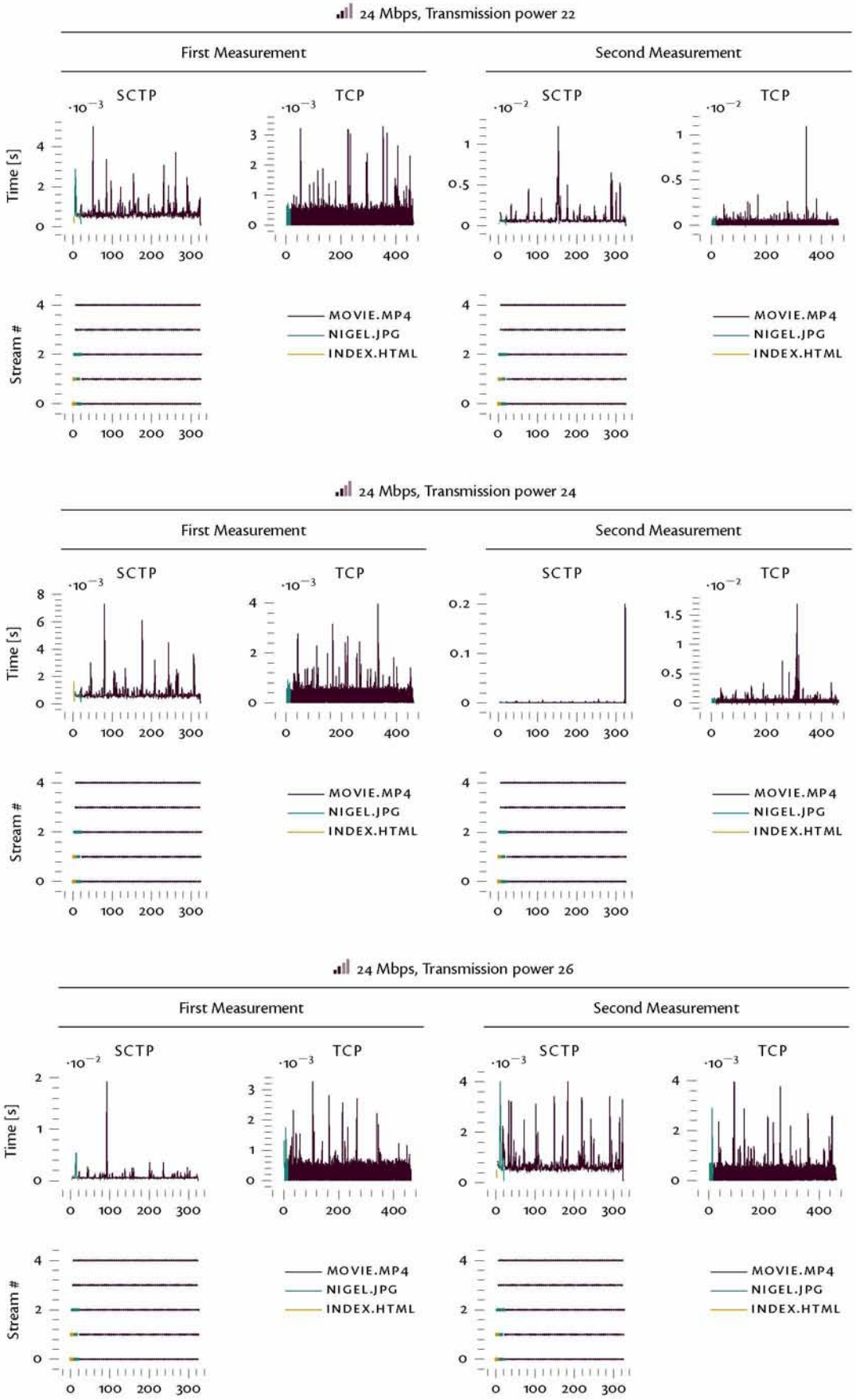


Figure A.25: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXV)

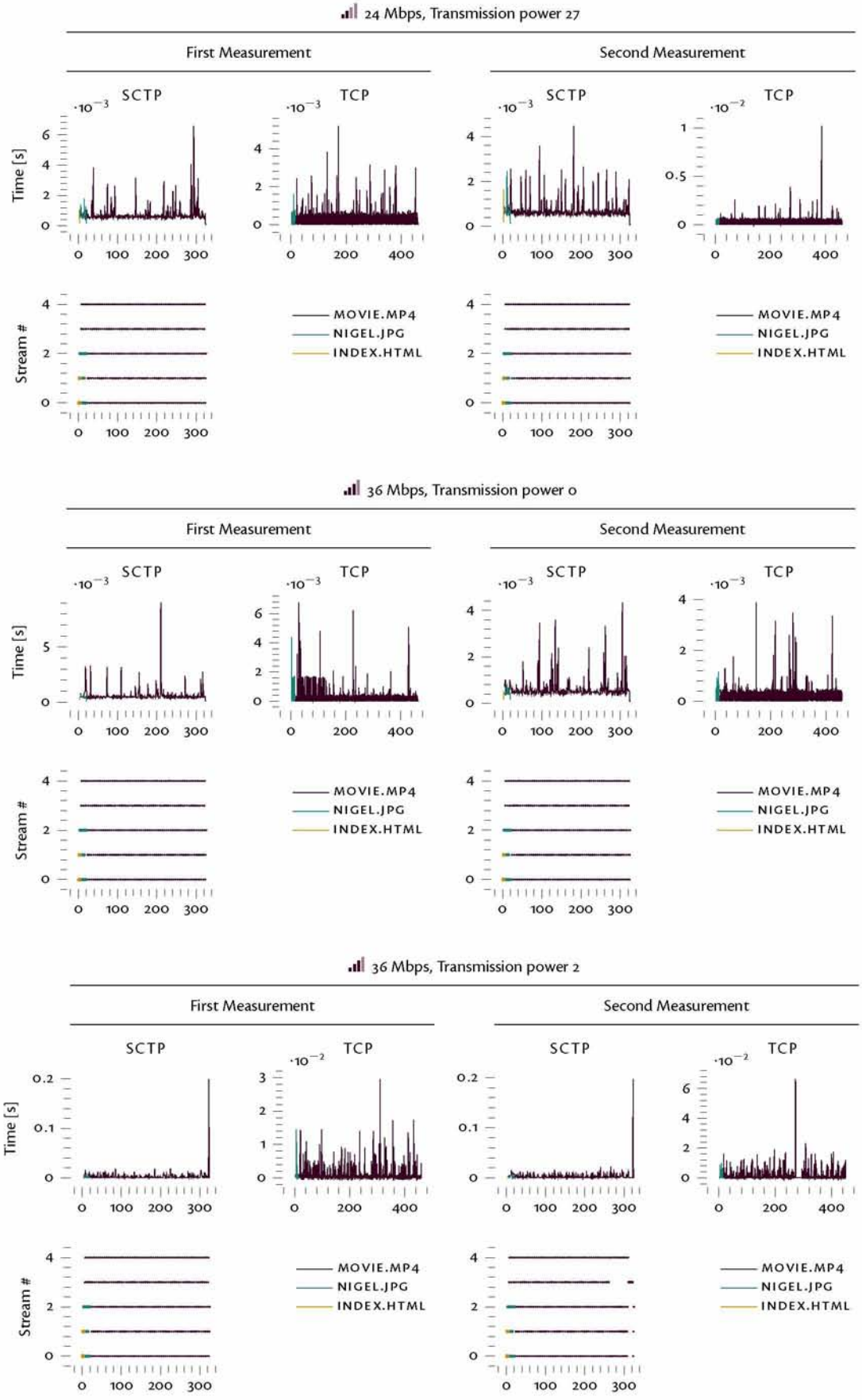


Figure A.26: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXVI)

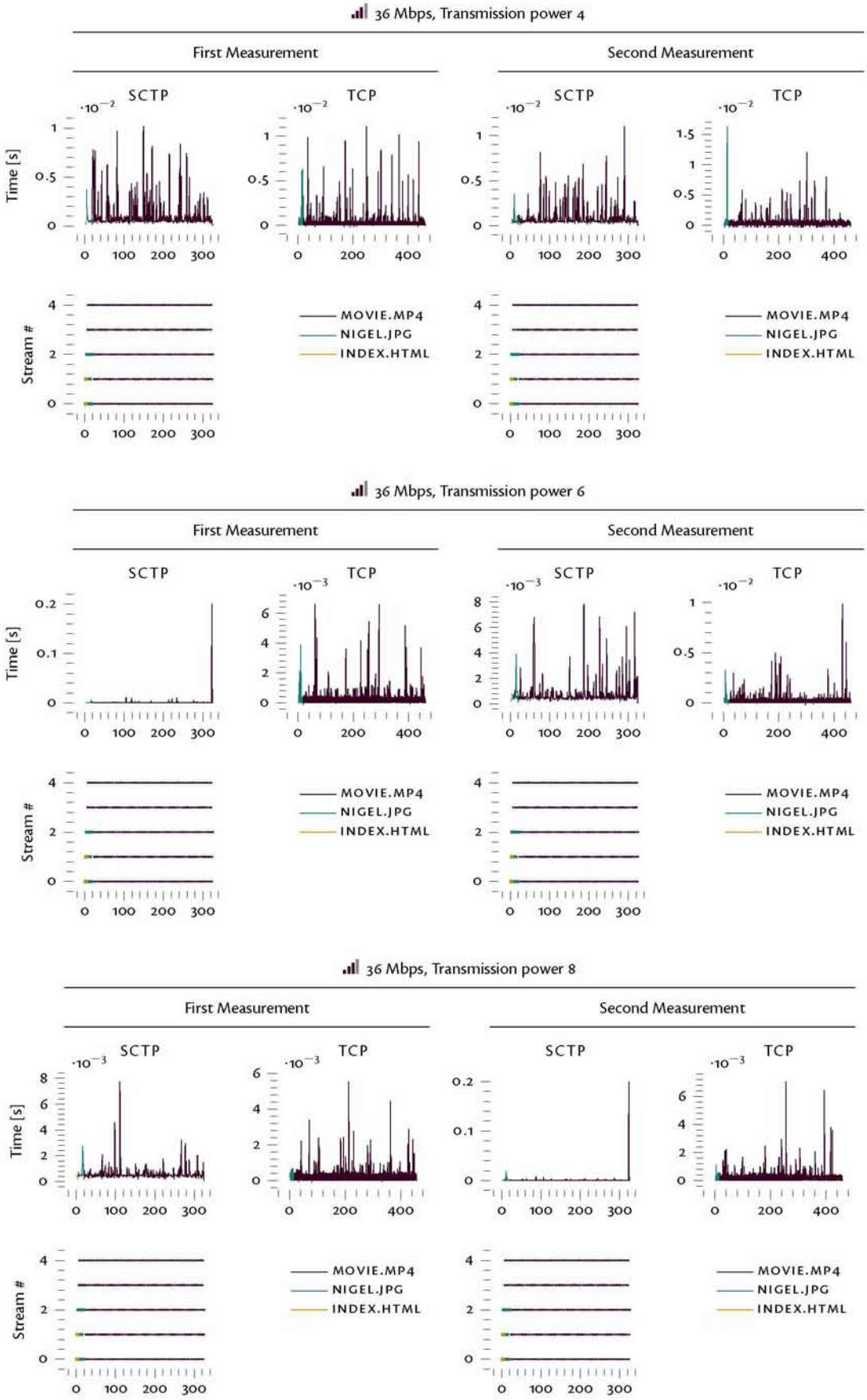


Figure A.27: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXVII)

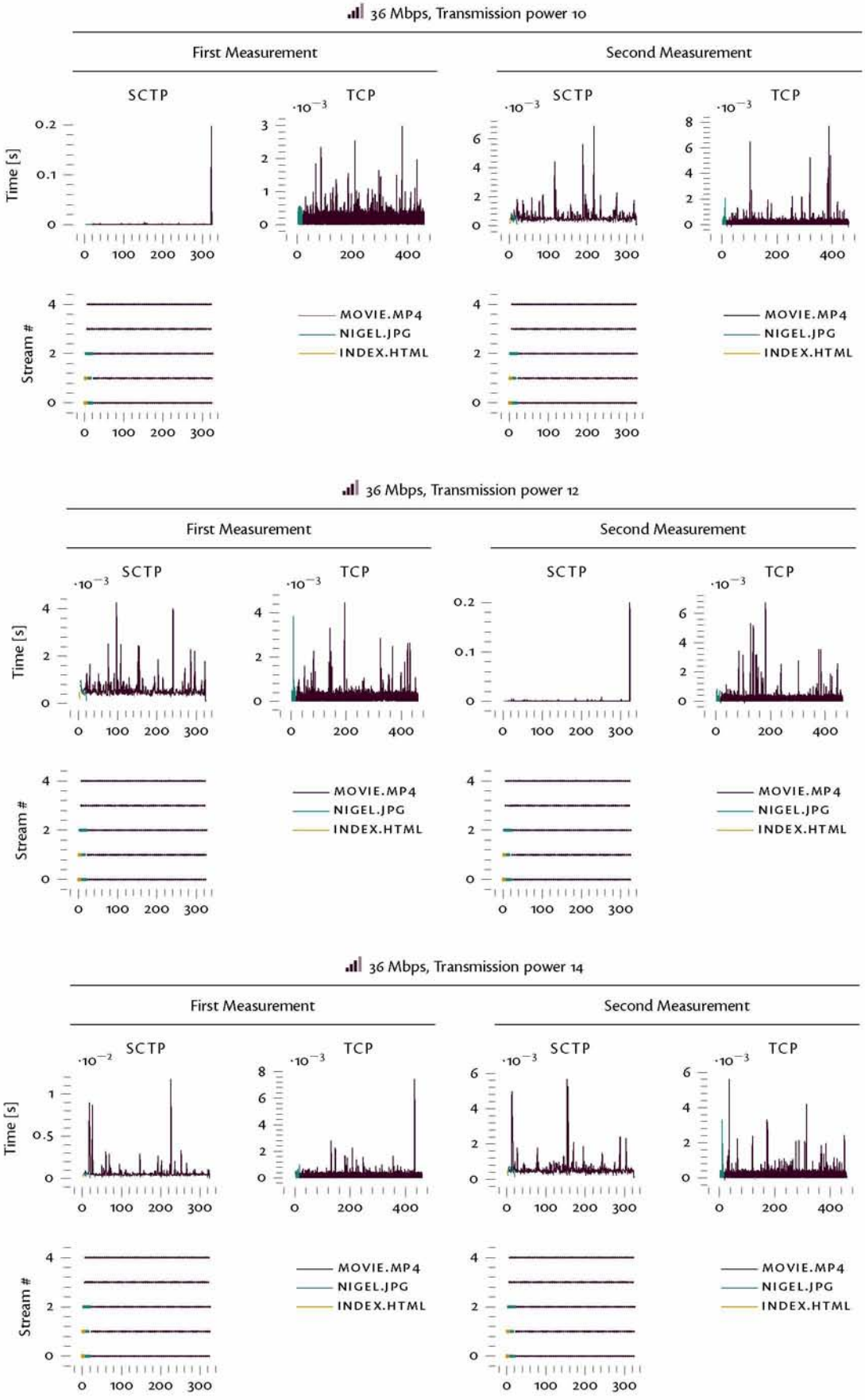


Figure A.28: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXVIII)

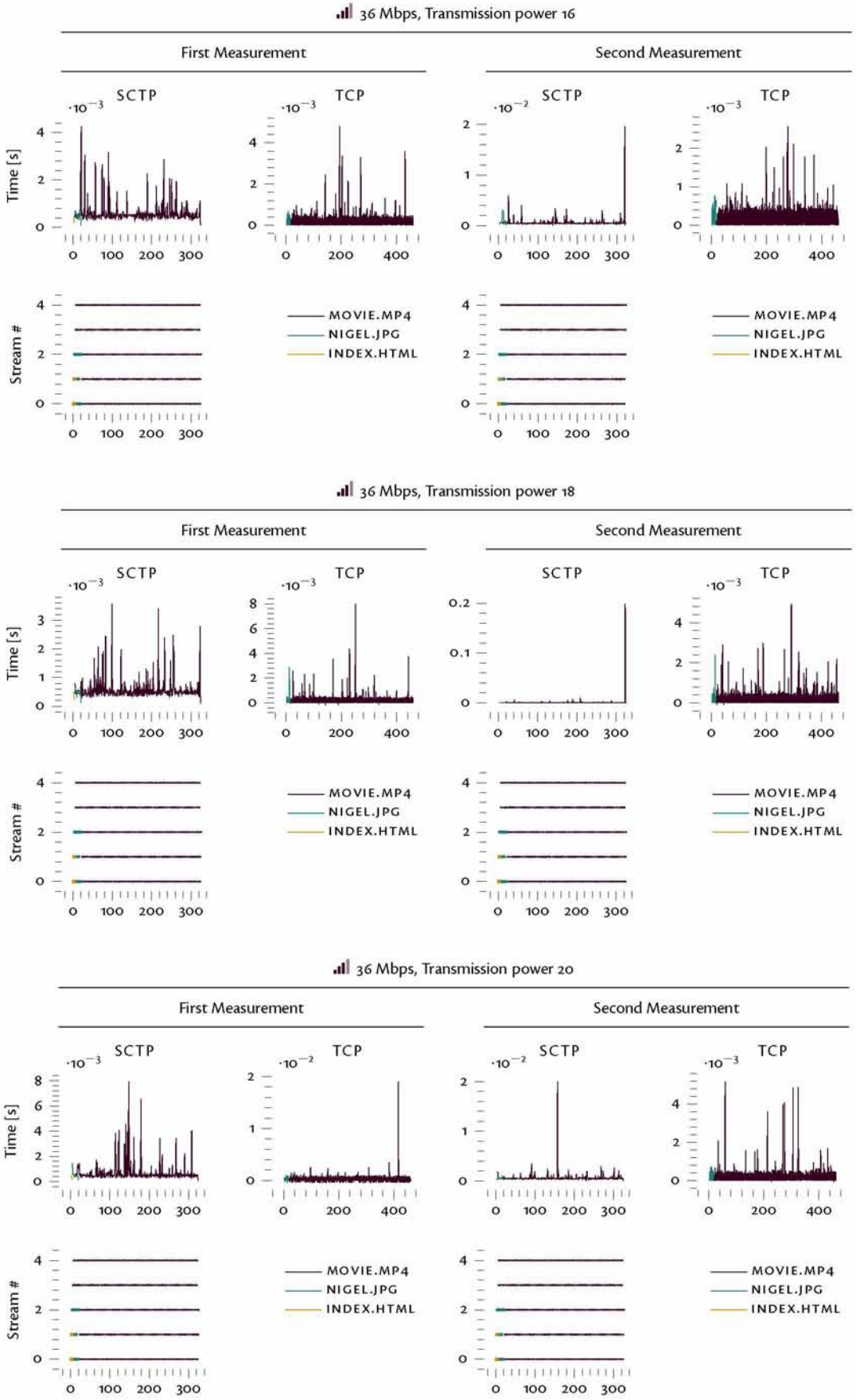


Figure A.29: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXIX)

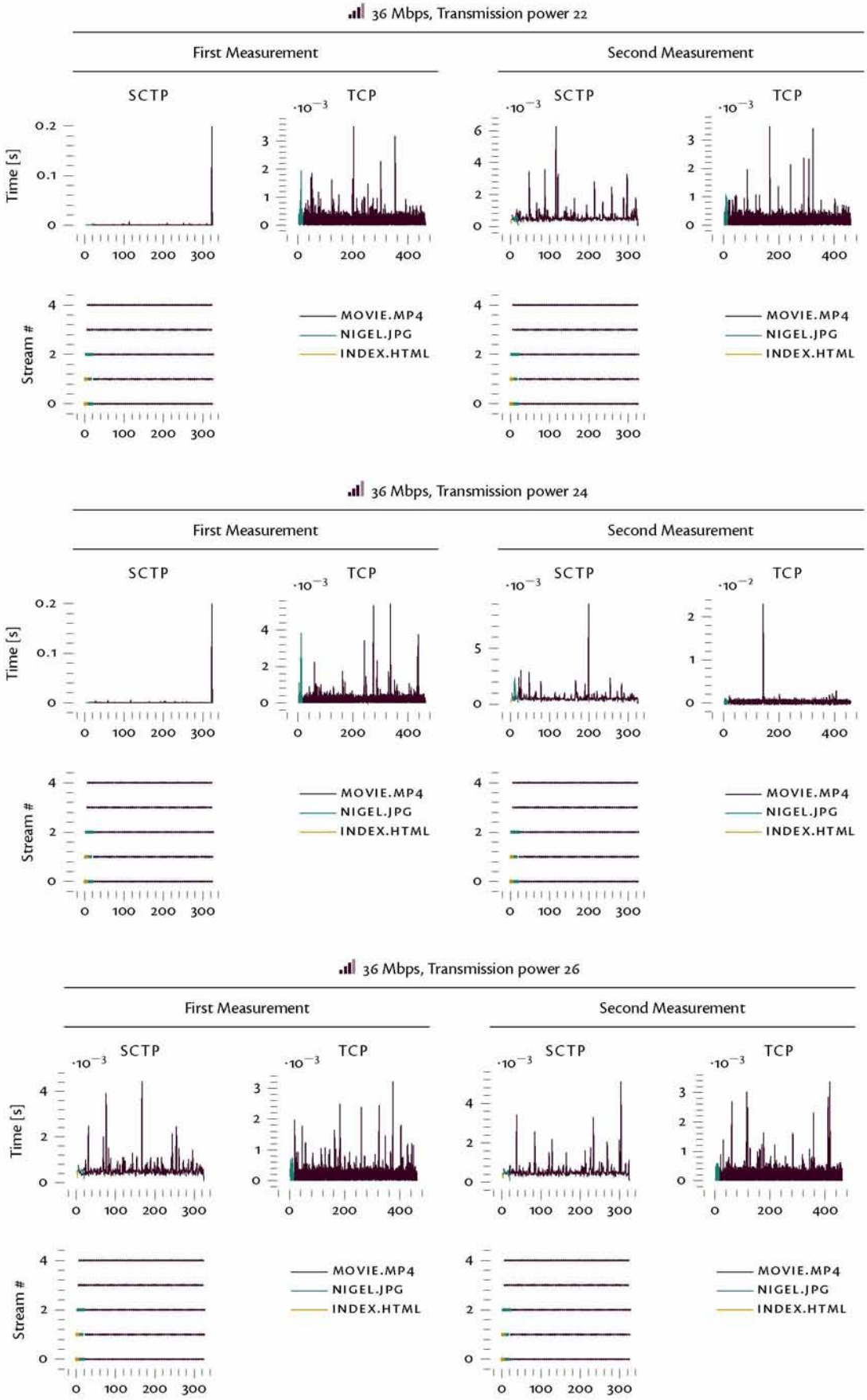


Figure A.30: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXX)

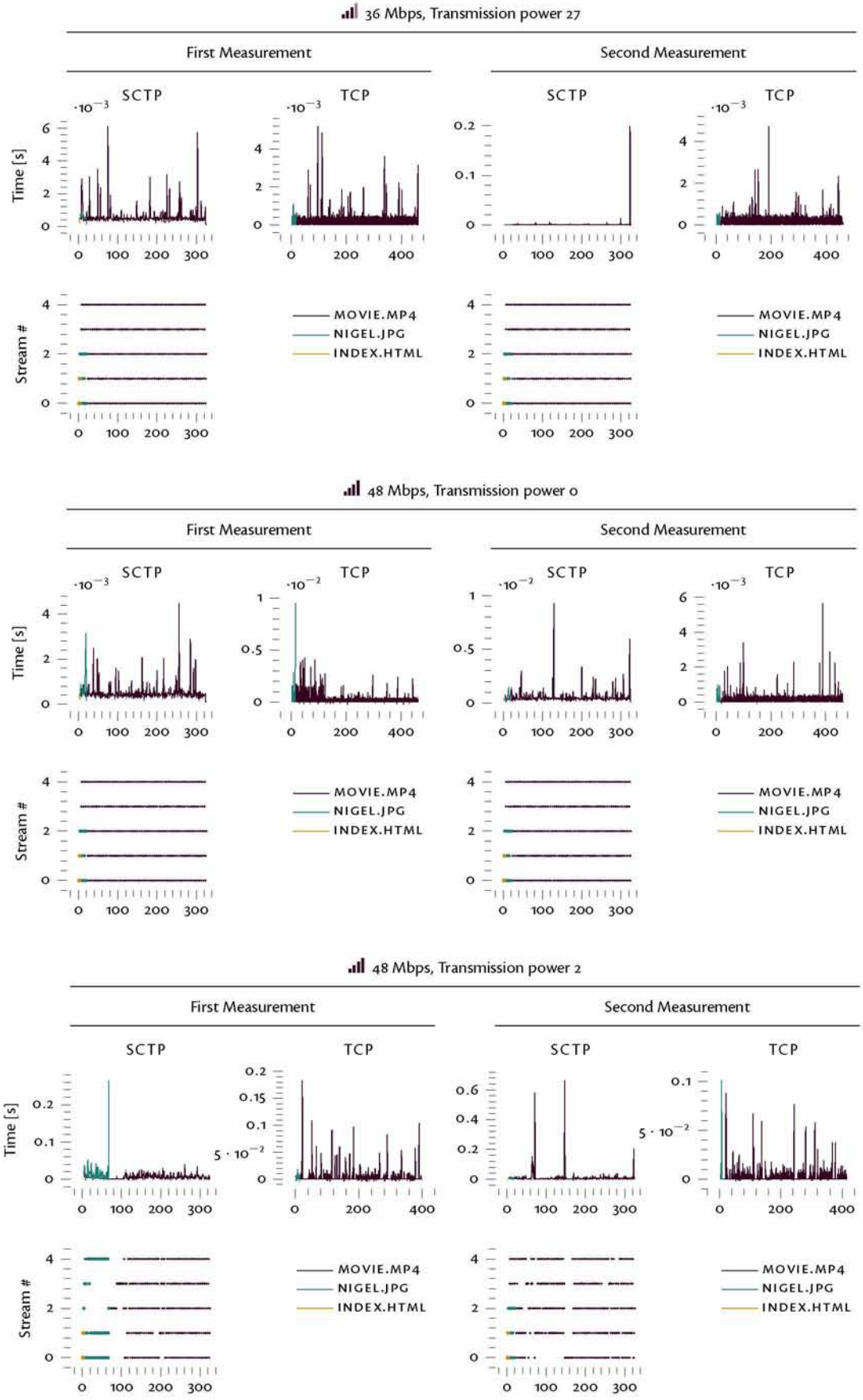


Figure A.31: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXI)

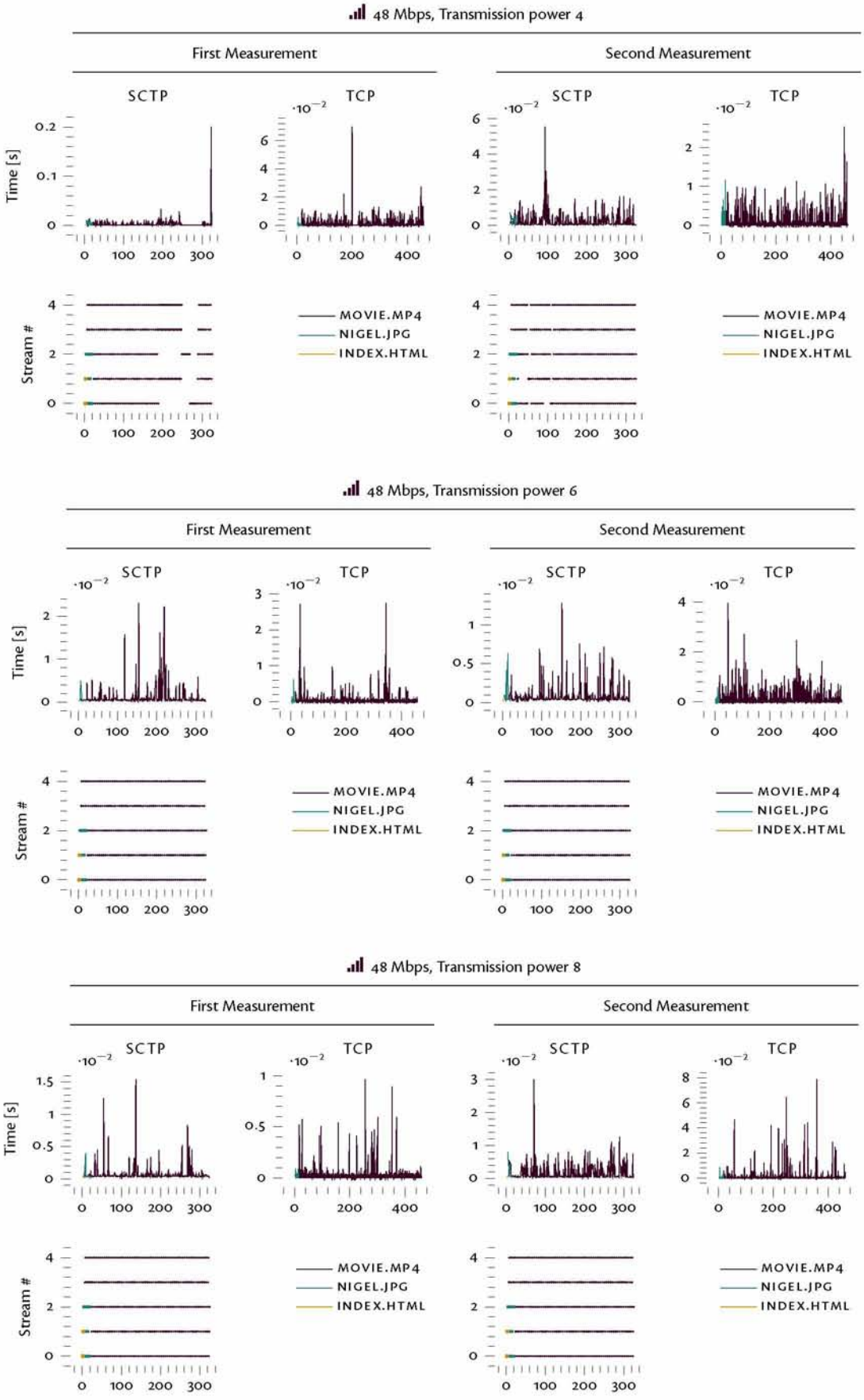


Figure A.32: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXII)

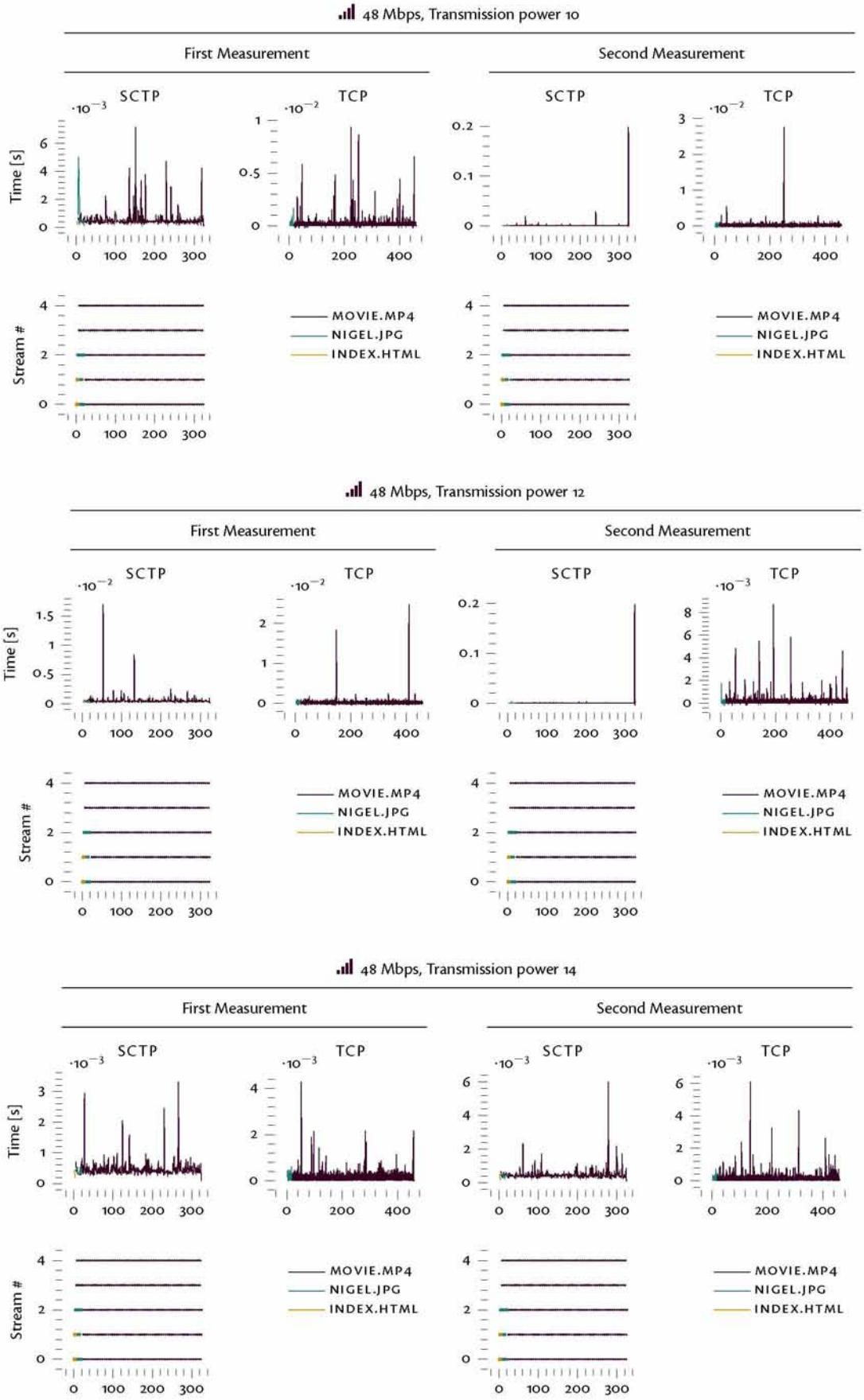


Figure A.33: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXIII)

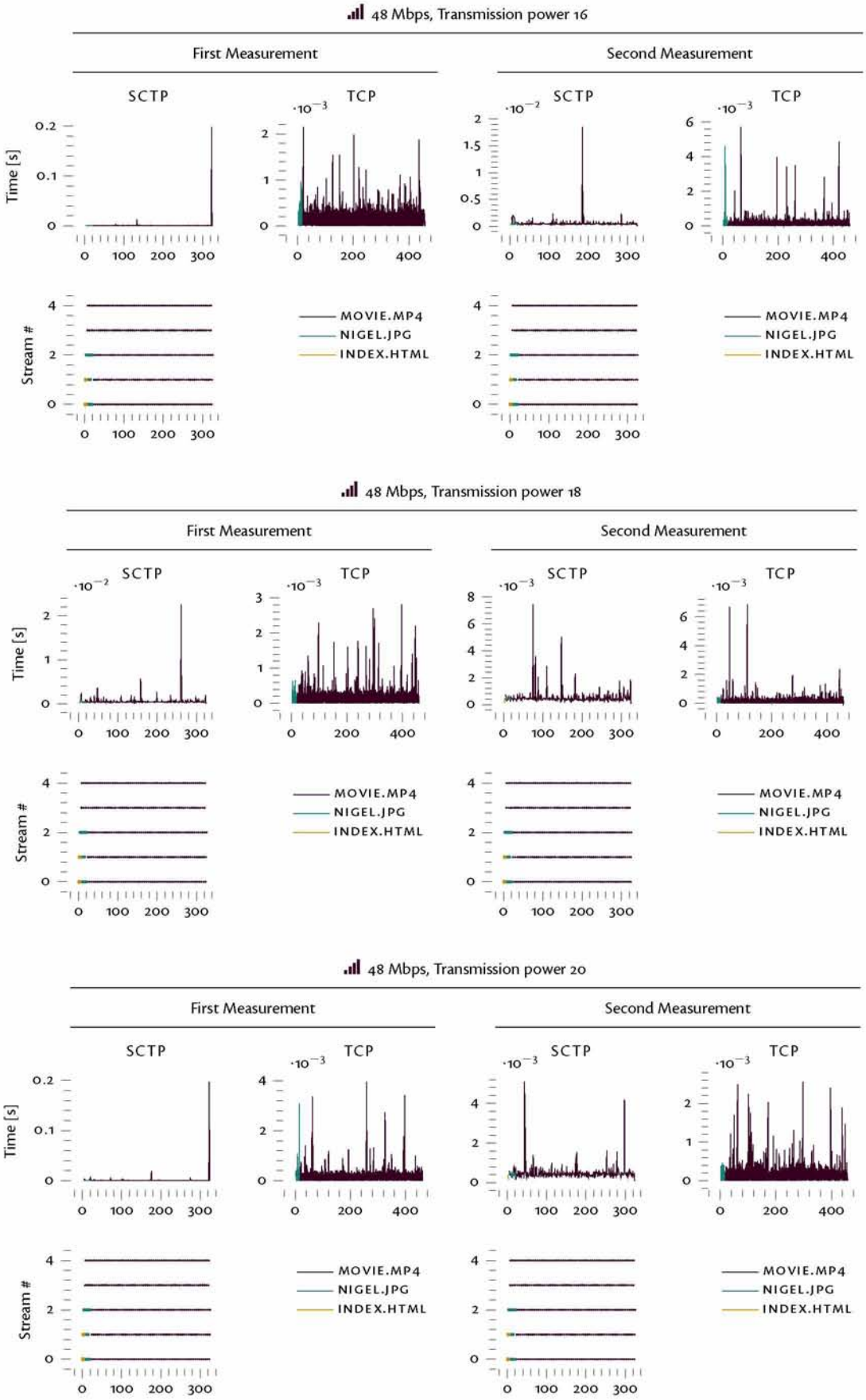


Figure A.34: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXIV)

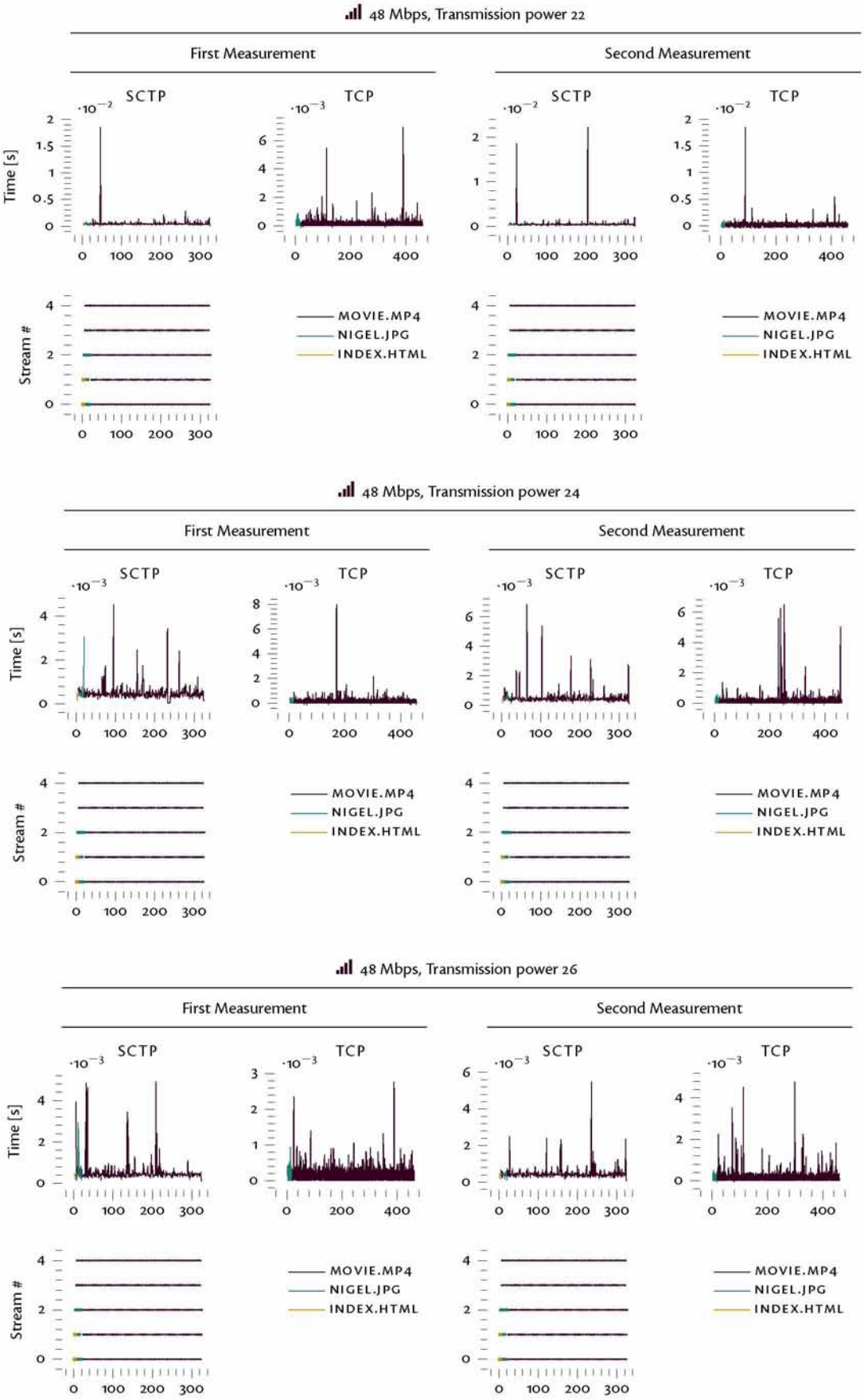


Figure A.35: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXV)

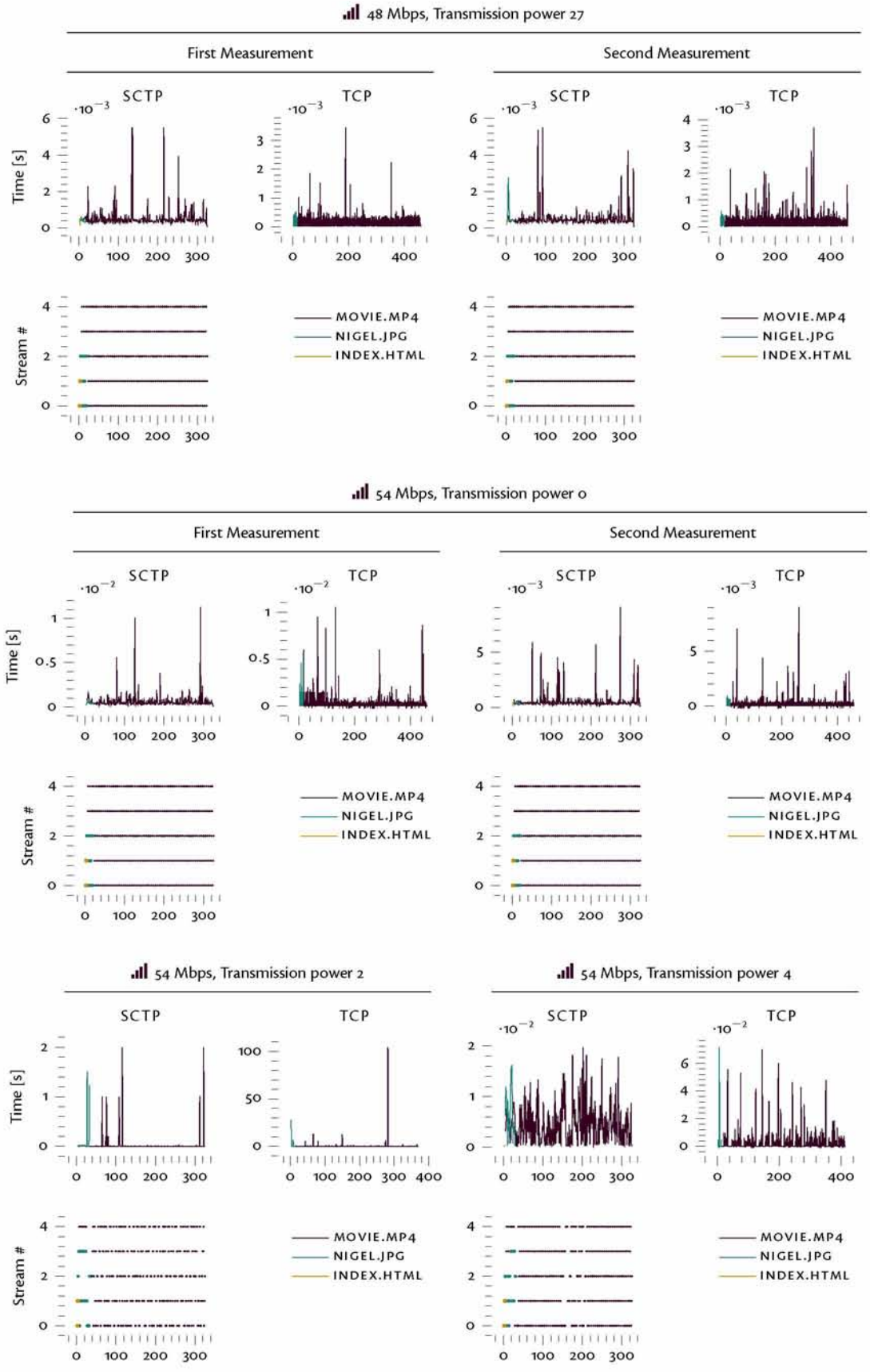


Figure A.36: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXVI)

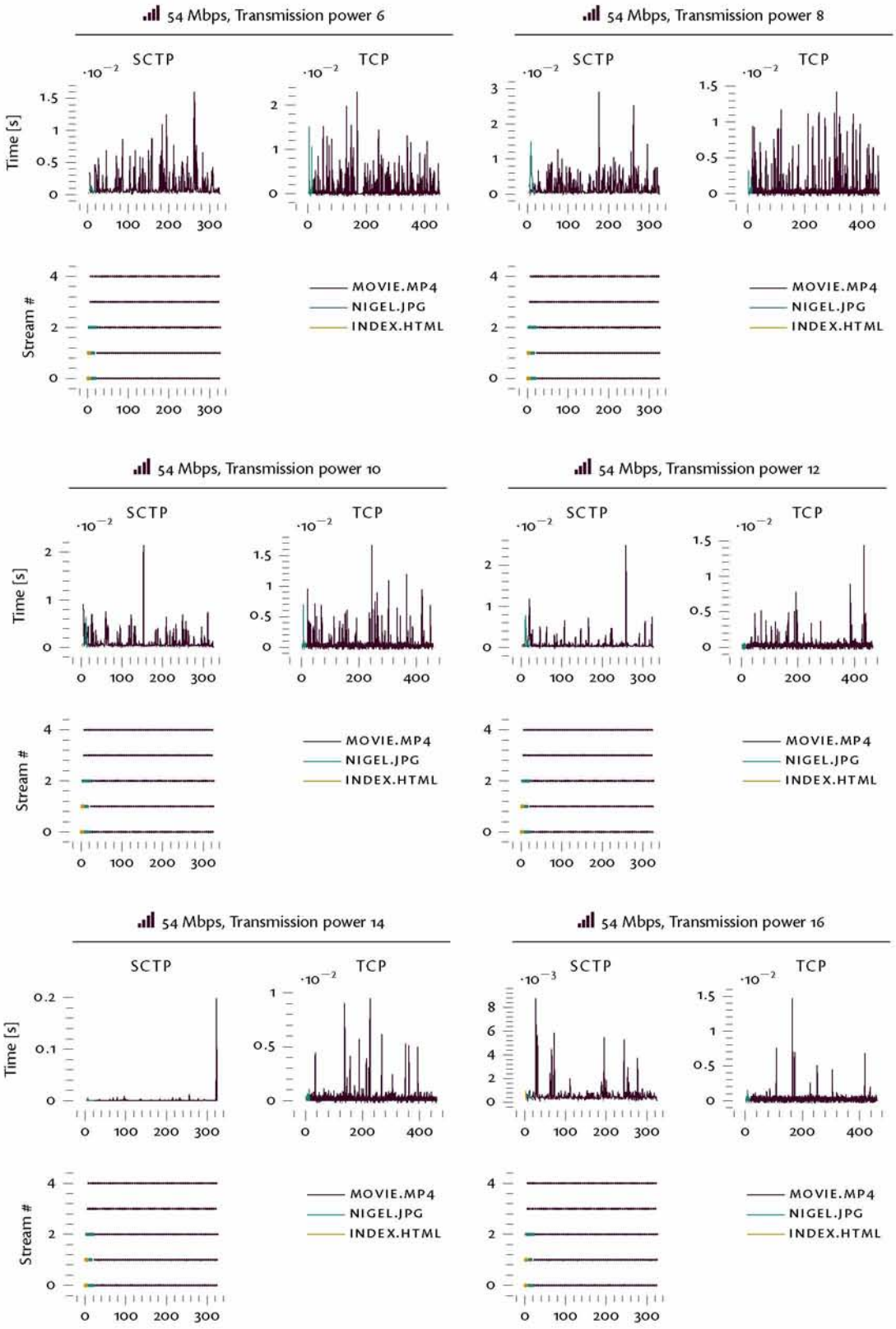


Figure A.37: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXVII)

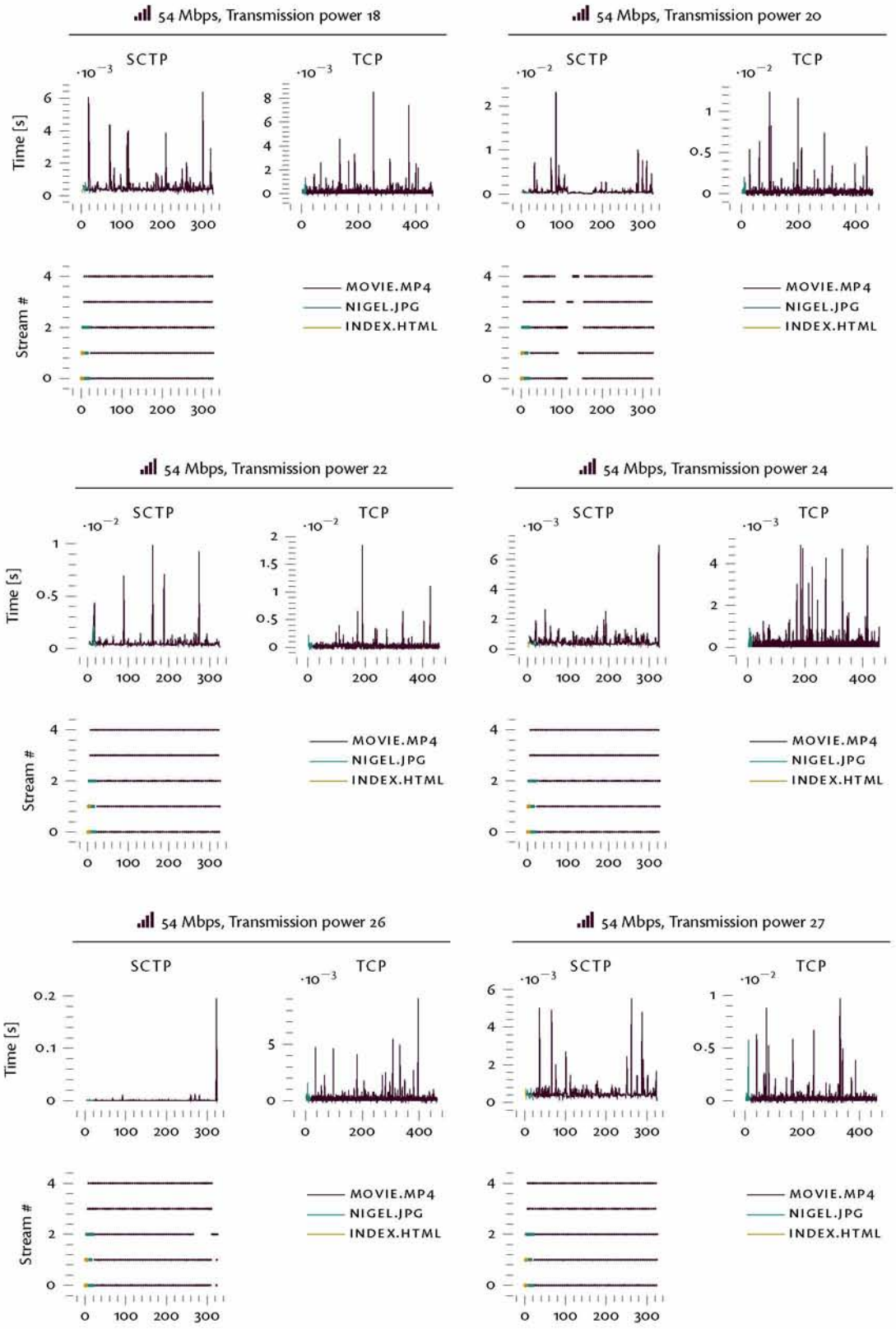


Figure A.38: Experimental measurements plots for the Wi-Fi 802.11g protocol (part XXXVIII)

A.2 Wi-Fi 802.11a

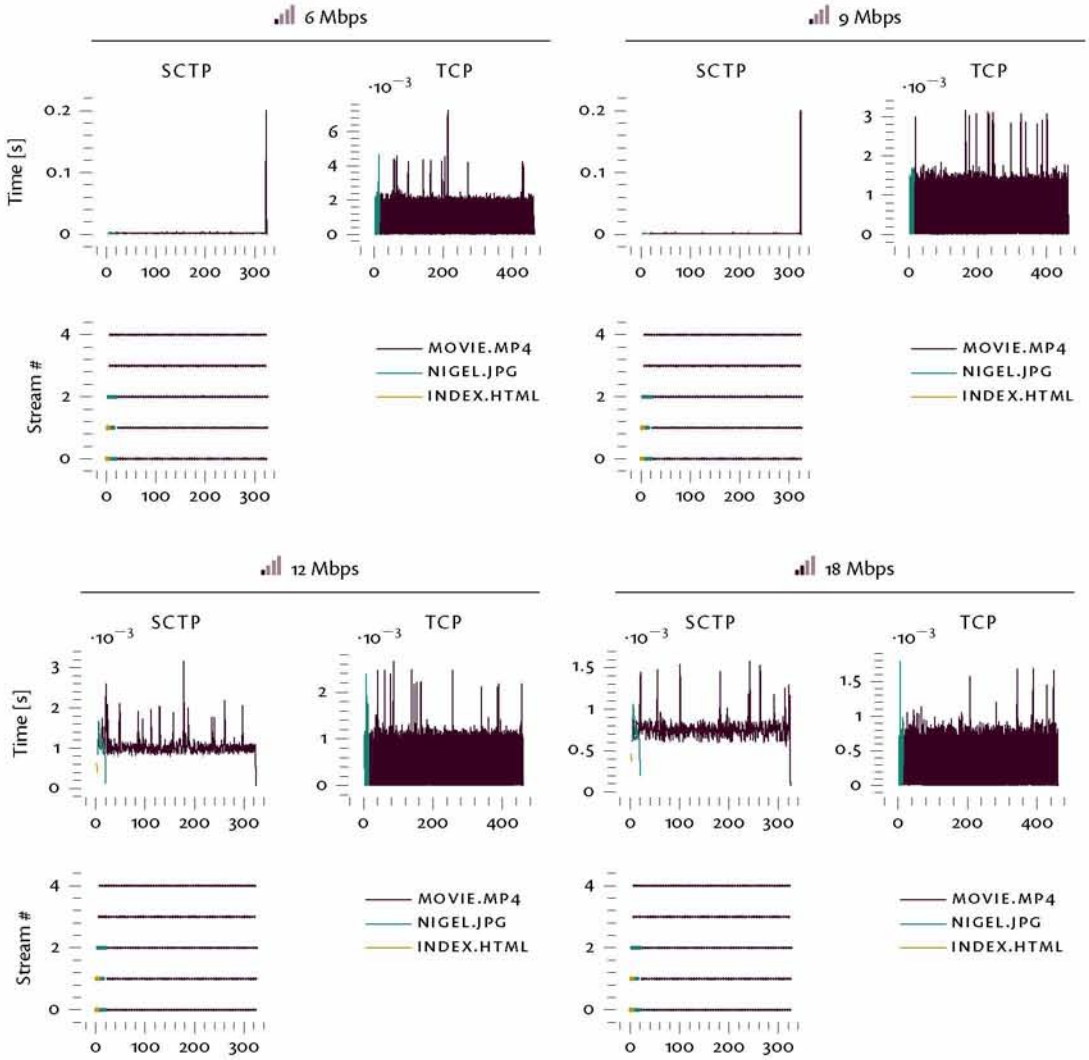


Figure A.39: Experimental measurements plots for the Wi-Fi 802.11a protocol (part I)

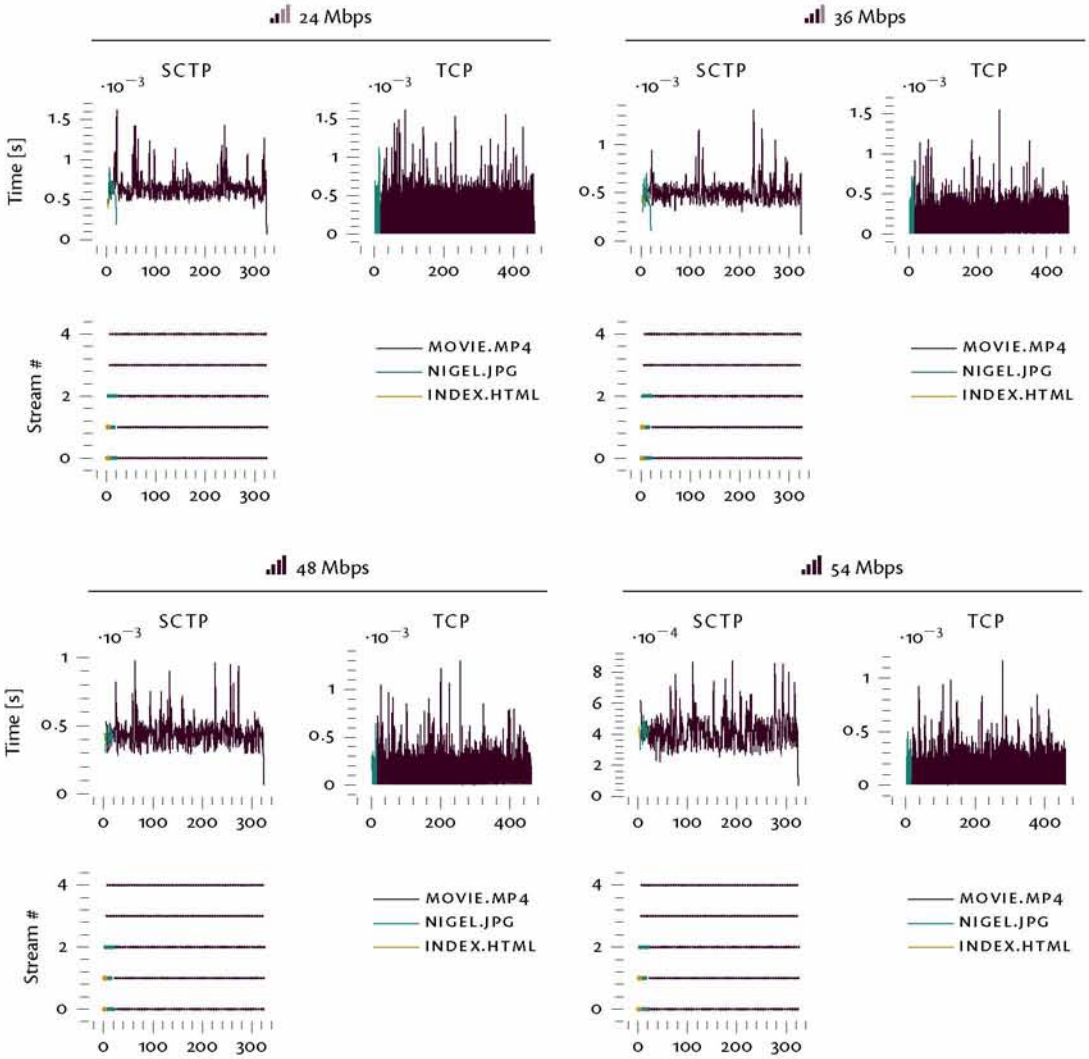


Figure A.40: Experimental measurements plots for the Wi-Fi 802.11a protocol (part II)

A.3 WiMAX

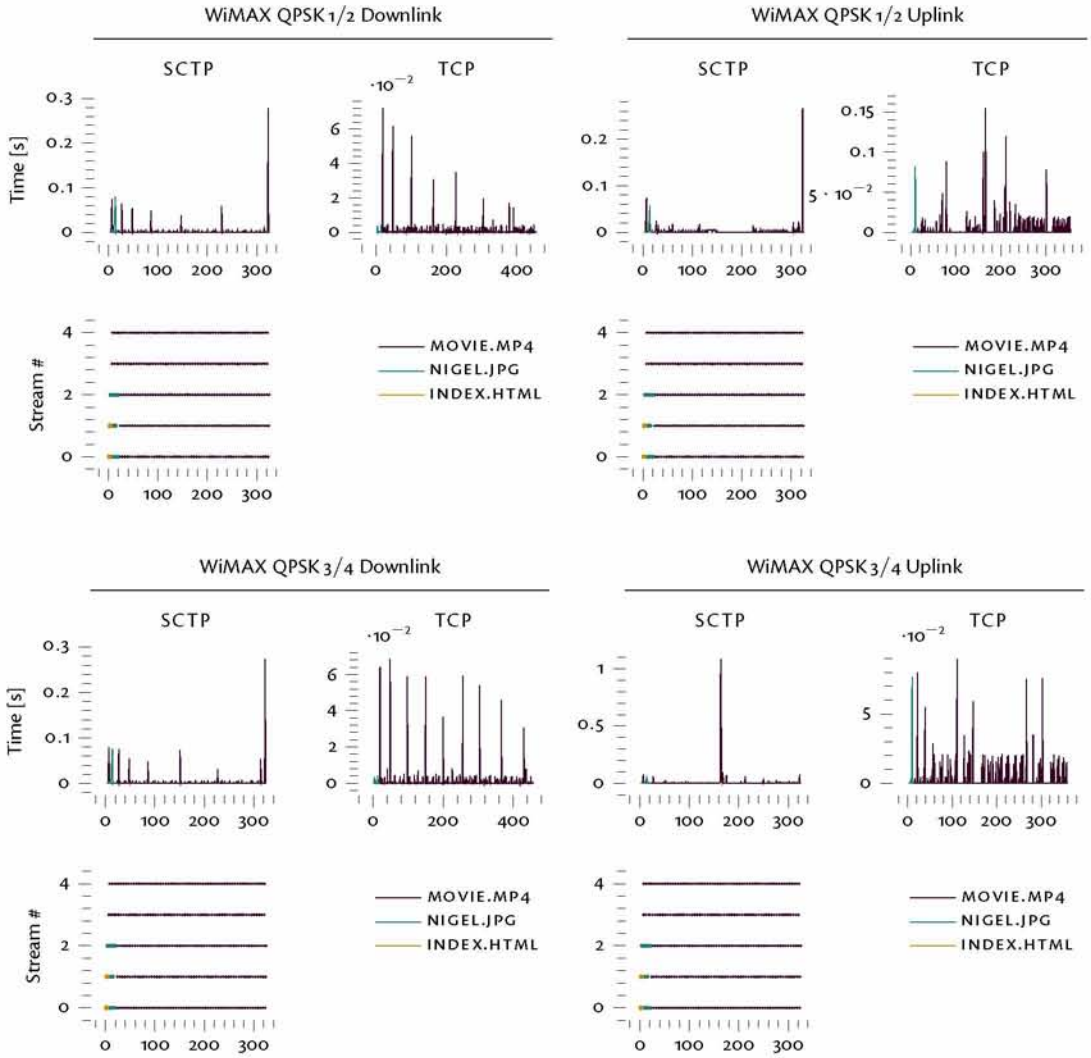


Figure A.41: Experimental measurements plots for the WiMAX QPSK

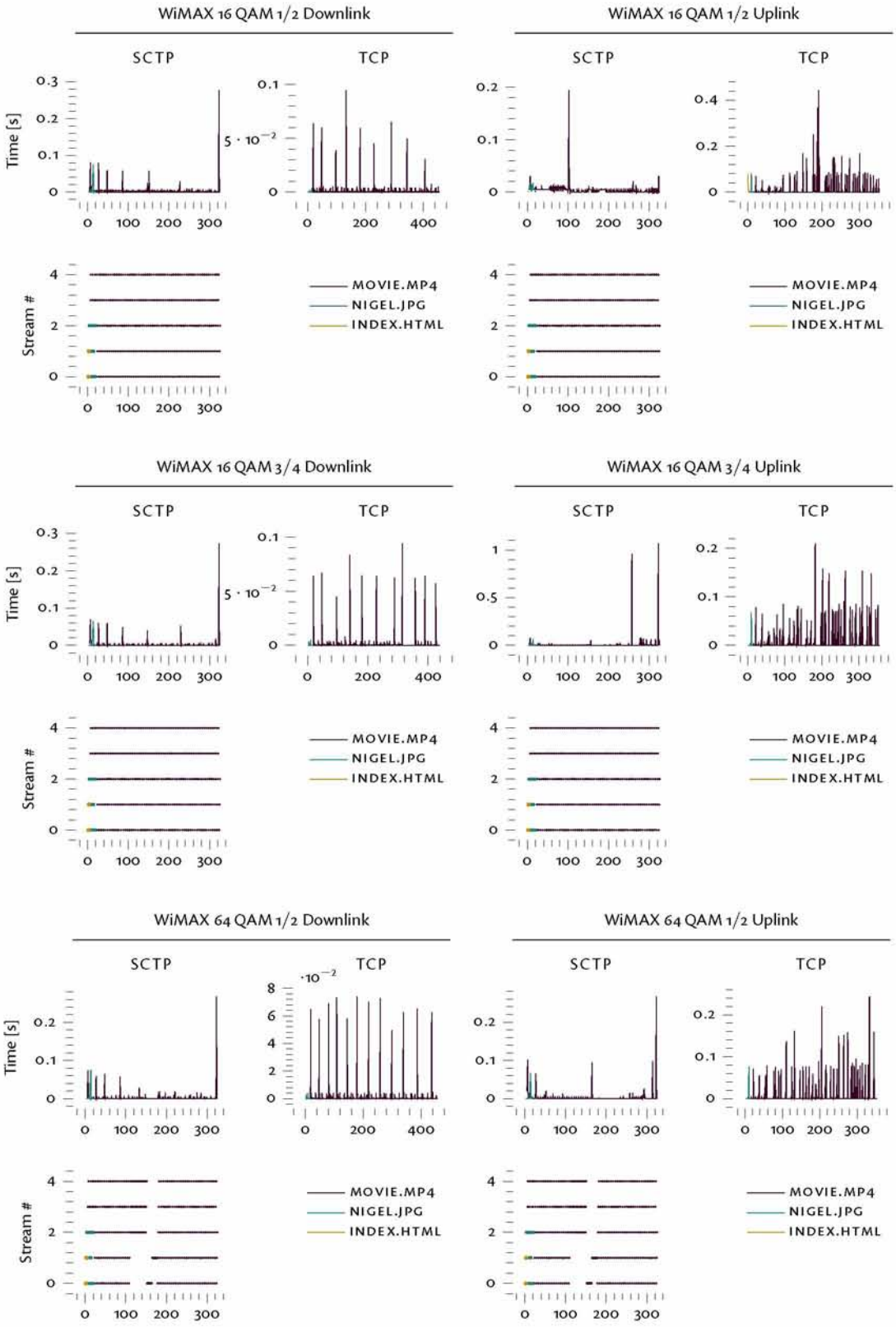


Figure A.42: Experimental measurements plots for the WiMAX QAM (part I)

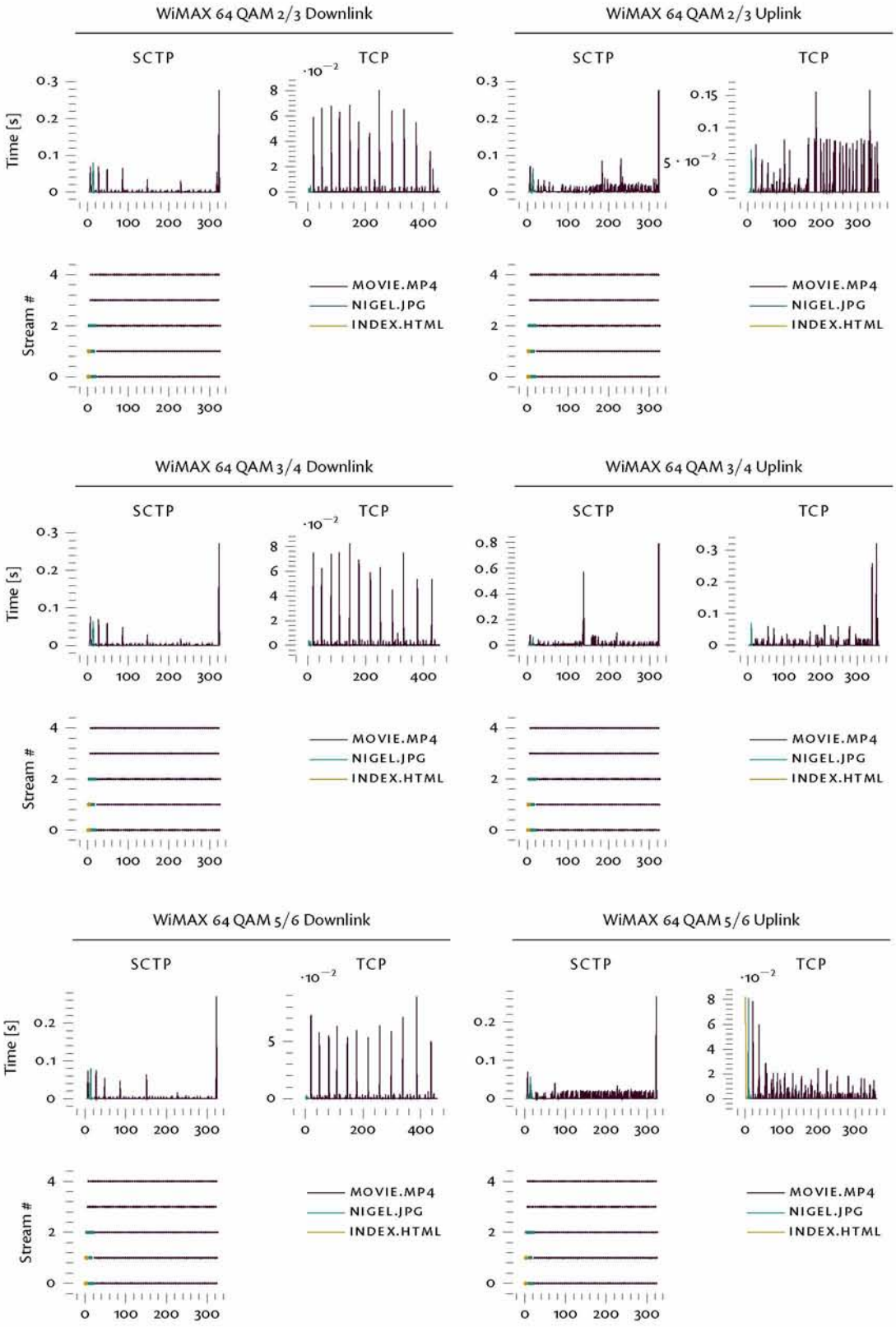


Figure A.43: Experimental measurements plots for the WiMAX QAM (part II)

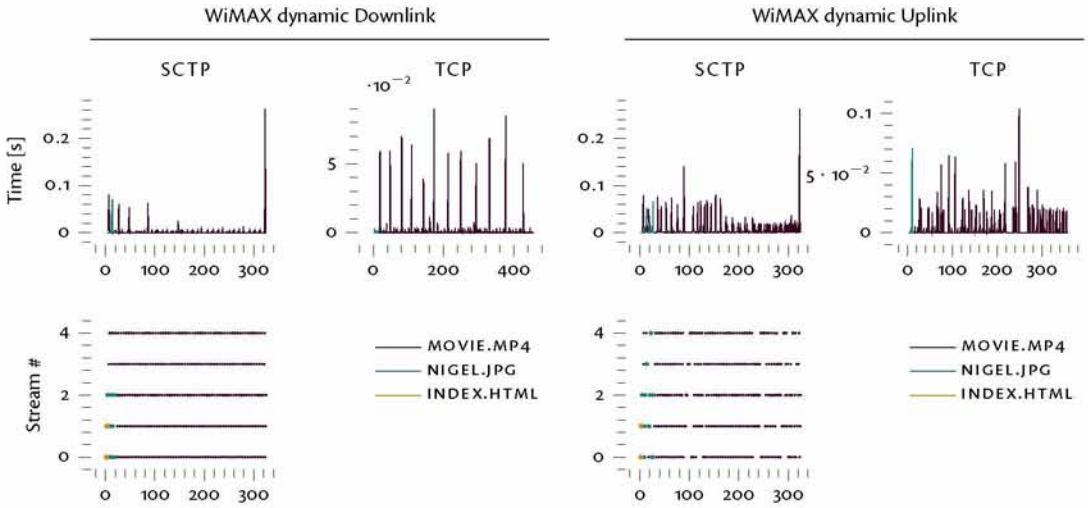


Figure A.44: Experimental measurements plots for the WiMAX (default MCS)