Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 3681270, 11 pages
https://doi.org/10.1155/2018/3681270

Research Article

A Novel UDT-Based Transfer Speed-Up Protocol for Fog Computing

Zhijie Han,[1] **Weibei Fan**,[2] **Jie Li**,[3] **and Miaoxin Xu**[1]

[1]*Institute of Data and Knowledge Engineering, Henan University, Kaifeng, China*
[2]*School of Computer Science and Technology, Soochow University, Suzhou, China*
[3]*Software College, Henan University, Kaifeng, Henan, China*

Correspondence should be addressed to Zhijie Han; hanzhijie@126.com

Received 24 January 2018; Revised 24 March 2018; Accepted 3 April 2018; Published 13 May 2018

Academic Editor: Xuyun Zhang

Fog computing is a distributed computing model as the middle layer between the cloud data center and the IoT device/sensor. It provides computing, network, and storage devices so that cloud based services can be closer to IOT devices and sensors. Cloud computing requires a lot of bandwidth, and the bandwidth of the wireless network is limited. In contrast, the amount of bandwidth required for "fog computing" is much less. In this paper, we improved a new protocol Peer Assistant UDT-Based Data Transfer Protocol (PaUDT), applied to Iot-Cloud computing. Furthermore, we compared the efficiency of the congestion control algorithm of UDT with the Adobe's Secure Real-Time Media Flow Protocol (RTMFP), based on UDP completely at the transport layer. At last, we built an evaluation model of UDT in RTT and bit error ratio which describes the performance. The theoretical analysis and experiment result have shown that UDT has good performance in IoT-Cloud computing.

1. Introduction

Internet of Things (IoT) and cloud computing are two very different technologies that are both already part of our life. IoT has received attentions for years and is considered as the future of Internet and also recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries [1–3]. Despite the increasing usage of cloud computing, there are still issues unsolved due to the inherent problem of cloud computing such as unreliable latency, lack of mobility support, and location-awareness [4]. However, cloud computing is considered as a promising computing paradigm due to the limited computation/storage on smart devices, which can provide elastic resources to applications on those devices.

Fog computing is proposed to enable computing directly at the edge of the network, which satisfies the strongly requirements in processing big data influx in real-time and functioning the available bandwidth bounds. Fog computing is an extension of cloud computing to the emerging IoT. Fog computing is not made up of powerful servers but consisted of a variety of functional computers with weaker performance and more dispersive performance. It is connected to a variety of devices, including mobile phones, wearable devices, smart TV, smart homes, smart cars, and even smart cities. It is a novel paradigm realizing distributed computing, network services, and storage from beyond cloud computing data centers up until the devices. At present, the amount of equipment to collect data and the amount of data processing are increasing exponentially. Public cloud computing provides computing space for processing the data through a remote server. However, after uploading these data to remote servers for analysis, it will take time to transfer the results to the original location, which will slow down the process of real-time quick response. Fog computing is a newly introduced concept that aims to put the cloud closer to the end users (things) for better quality of service [5, 6].

The transmission control protocol (TCP), the de facto transport protocol of the Internet, substantially underutilizes network bandwidth over high-speed connections with long delays [7]. Although the transmission control protocol (TCP) plays a major role in the Iot-cloud today, TCP does not suit the high network bandwidth-delay product (BDP) because of its congestion control algorithms. UDT is a

UDP-based approach, to the best of our knowledge, it is the only UDP-based protocol that employs a congestion control algorithm targeting shared networks. Compared to transmission control protocol (TCP), UDP does not yield retransmission delay, which makes it attractive to delay sensitive applications. In Iot-Cloud, most of all, the high performance of the congestion control of TCP is composed of four core algorithms: slow start, congestion voidance, fast retransmit, and fast recovery. In order to prevent network congestion where the congestion window size (CWnd) grows too fast, we need to set a threshold (ssthresh) of slow start state. With the changing of CWnd, different algorithms are used in different scene. Gu and Grossman [8] proposed the UDT protocol, which is an improvement on UDP. Considering the characteristics of the fog computing that concentrates data, data processing, and application on the edge of the network, we improve the UDT protocol, which can be better applied to the data transmission of fog computation.

With the size and complexity of cloud computing expanding, the architecture pattern is becoming more crucial on the performance of cloud computing. Different functions of cloud computing have different requirements for architecture patterns. The traditional UDT protocol is based on the Client and Server (C/S) mode, which is the most common distributed computing model [9]. The C/S model is very suitable for the situation where resources are relatively centralized. It is also easier to implement, but it may cause a slow response to all clients when the amount of access becomes larger. In modern data centers, the adoption of more high-speed Ethernet uplink has been the mainstream. The host (server) connection has been transferred from the 1G network to the 10G network and continues to evolve towards the 25G network. The connection between switches will evolve from 10G to 40G and even 100G. Just as the data center of Facebook is built [10], every 10G bandwidth top switch is connected to the optical fiber switch with a 40G uplink and then access the underlying server. The traffic inside the data center is a huge number. Its scale is usually 1000 times the outflow of the data center, and more and more data centers are experiencing the change of data transmission speed.

Peer-to-peer (P2P) is a mature network that share computer resources and services through direct exchange of information between network nodes [11]. The P2P architecture eliminates the central position of the server, such that servers can share computer resources and services directly in each system through exchange. The concept of fog computing itself does not clearly define the location and distribution of the calculation, and this is not the main problem to be solved in the fog computing. In order to take advantage of the increasing computing power of personal devices, the concept of P2P cloud is proposed [12]. We improve the UDT protocol with combining the characteristics of fog computing and P2P network and propose a UDT-based transfer speed-up protocol for fog computing.

The rest of the paper is organized as follows. Section 2 gives an overview of the UDT protocol. Section 3 presents an overview of the UDT and describes its design and implementation. Section 4 gives a comparison between RTMFP and UDT and gives an experimental evaluation of the UDT performance. Section 5 concludes the paper.

2. Related Work

New network devices are emerging and starting to connect. The data generated and sent to the cloud will increase exponentially. If storage and computing power follow Moore's law, it means that they will double every 18 months. The relative speed of bandwidth seems to be much slower. Some research institutions estimate that the global bandwidth rate is growing less than 40% per year. This means that there will be more data to be sent to the cloud, but it will be constrained by bandwidth speed.

As a consequence, real-time and latency-sensitive computation service requests to be responded by the distant Cloud data centers often endure large round-trip delay, network congestion, service quality degradation, etc [13]. A new transport protocol as a timely solution is required to address this challenge. With IoT becoming a major component of fog computing, it is becoming more and more important to improve quality of service (QoS) in fog computing networks. UDT is a UDP-based protocol that proposed by Gu and Grossman [8], which is a radical change by introducing a new transport layer protocol involving changes in routers. The new protocol is expected to be easily deployed and easily integrated with the applications, in addition to utilizing the bandwidth efficiently and fairly.

A lot of researches have been carried out by this protocol. Currently most interaction between the IoT devices and the supporting back-end servers is done through large scale cloud data centers, location awareness [14], and widespread geographical distribution for the IoT. Liu et al. presented analytical results based on UDT that highlight transport characteristics specific to dedicated connections and established the monotonicity, concavity, and stability of the throughput profiles under various configurations. Park et al. [15] proposed a congestion degree based MTR estimation algorithm, which tried to classify more depending on the congestion degree to estimate more actual available bandwidth. Their proposed method showed that the fairness problem among the competing flows is significantly resolved in comparison with that of UDT. In [16], Park et al. also proposed a congestion degree based MTR estimation method, which can be preemptively estimated compared with that of round trip delay information on sender side. The simulation results showed that the competing flows with the proposed method can fairly occupy the network bandwidth. Murata et al. [17] proposed a novel transport protocol, named high-performance and flexible protocol (HpFP), to show high throughputs for the HbVRS in LFNs with packet loss. They examine basic performances of multiple streams of the HpFP and the UDT on a laboratory experiment simulating an international LFN.

As main concepts of cloud computing, fog provides service which includes computation, storage, and networking services to end users, but in the end of the network. Despite the countless benefits of fog, the research in this field is still immature, and many researchers still are working on defining

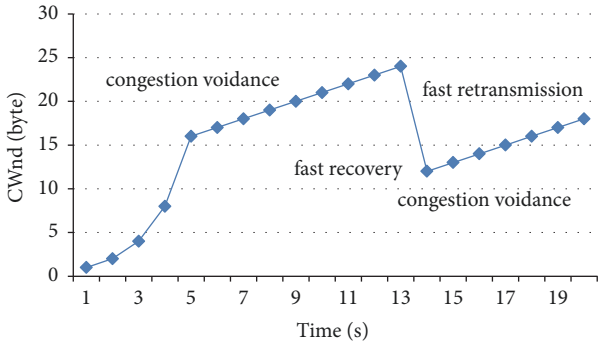

Figure 1: The congestion control algorithm of TCP.

vision, basic notions, and challenges of fog computing [4, 14, 18, 19]. Bernardo and Hoang [20] outlined security requirements for UDT implementation and proposed practical encryption methods for securing UDT within the network layer.

An open research challenge is to explore the potential benefits of fog computing and IoT. In other words, one must study that quality of transparent service will be improved by having a layer of fog nodes between IoT and the cloud. Recent work addressed the design of TCP or UDP for transmission from mobile subscribers to edge clouds, to achieve a power-delay trade-off. Despite their solid contributions, the proposed approach is limited to the cellular network infrastructures. It assumes an IoT device can be only a User Equipment (UE) and that edge servers must be attached to base stations. Additionally, by not considering the cloud in the approach, scenarios where IoT-cloud or fog-cloud communication takes place are not handled.

3. Improved UDT Protocol for Fog Computing

For fast retransmission and fast recovery in IoT network, when an Iot client which we called sender receives three Native Acknowledgements (NAKs), sender will retransmit the lost packets right away and cut the ssthresh by half. At the same time, the algorithm sets congestion window CWnd equal to ssthresh and congestion voidance starts. The whole process is shown in Figure 1.

In Figure 1, the initial window size is 16, when the CWnd equals 16, the slow start stops, and the congestion voidance begins. In the stage of congestion voidance, the increase of CWnd is additive. As shown in Figure 1, when CWnd equals 24, network starts blocking, and at this time, the ssthresh is set to 12 (half of 24).

The window based AIMD (additive increase multiplicative decrease) control algorithm of TCP suffers from random loss as the BDP increases to higher [21, 22]. In order to resolve the TCP's inefficiency problem in high BDP links, we use a new protocol, called UDT. UDT is an application level program and, in transport layer, it uses UDP to transmit data. UDT is a reliability and security streaming data transport protocol which is end-to-end, connection oriented, unicast, and duplex.

As illustrated in Figure 1, countless dividing lines exist between H_1 and H_2. The abscissa represents time and the ordinate represents the window size. For two-dimensional space, we want to find the best divider line. For the n-dimensional space, our ultimate goal is to find the best classification of the super-plane, that is, to find the final decision-making boundaries.

Table 1: UDT data packets header.

	Packet sequence number
F	Message number
Time stamp	
Destination socket ID	

3.1. UDT Structure. The UDT layer has five function components: the API module, the sender, the receiver, the listener, and the UDP channel. There are four data components: sender's protocol buffer, receiver's protocol buffer, sender's loss list, and receiver's loss list [6].

The API module is responsible for interacting with applications. When application sends data, the data packets are passed from sender's buffer to UDP channel. On the other hand, receiver reads data from UDP channel and the packets will be reordered in receivers buffer. Receiver will check receiver's loss list; if there are packets in receiver's loss list, receiver will send a NAK to sender and when sender receives a NAK, the sender's loss list will be updated, and sender sends this packet again until the receiver receives this packet. The main transmitting procedure is shown in Figure 2.

3.2. The Packets Type of UDT. UDT has two kinds of packets: the data packets and the control packets. The types of packets are distinguished by the first bit (flag bit) of the packet header, and when the flag is 0, it represents a data packet, and otherwise it represents a control packet. As shown in Table 1, a UDT data packet contains a packet-based sequence number, a message number, a destination socket id, and a relative timestamp. "*F*" represents the position of this packet in the flow as follows: "10" is the first packet, "01" is the last one, "11" represents that there is only one packet, and "00" is any packet in the middle. The next bit whose value is 0 represents that the message is sent out of order, and if the value is 1 in this position shows that the message is sent in order. The 32-bit time stamp is a relative value starting from the time when the connection is set up.

The control information is shown in Table 2. The comparison of control packets with data packets in type and revered field is illustrated in Table 1. There are 8 types in control packets, such as handshake, keep-alive, ACK, and NAK. The revered field defines a new type of control packet or adds new variables in an existing control package for a new congestion control algorithm.

3.3. PaUDT Protocol Mode. In the structured P2P model, all information is scattered in different nodes of the network in the form of Hash list, and a huge distributed hash table (DHT) is constructed by the whole network (Figure 4) [23]. The whole network information is stored and retrieved distributed in the application layer. Kademlia is a typical

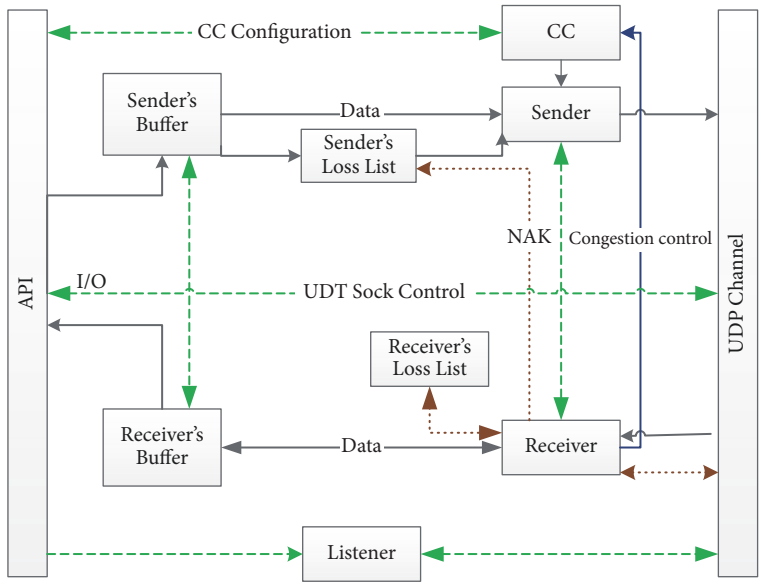

Figure 2: UDT/CCC implementation.

Table 2: UDT control packets header.

Type	Revered
Additional info	
Time stamp	
Destination socket ID	
Control information field	

structured model, which uses "XOR" to measure the distance between network nodes. Such a measure does not affect the efficiency and scalability of the network and provides better fault tolerance and flexibility. Consistent hash functions are widely used in distributed data storage, which guarantees dynamic changes in the distributed environment. The mapping space of the Hash function will be slowly changed and still maintain the load balance, thus ensuring the availability and efficiency of the system. The P2P network is also a dynamically changing distributed system, where nodes continue to join or leave the hash space in constant change. It relies on the consistency hash function to ensure the efficiency and load balancing of its location, query, data storage, and replication.

DHT technology is to add a separate DHT layer between the P2P network application layer and the network layer to locate and locate the resources of the P2P network. See Figure 3. DHT uses the Hash function to speed up the lookup, improve security, and make management convenient, without taking up too much of the network bandwidth. Kademlia is a typical structured P2P model, which assigns a unique and random node ID to each node. Each object assigns a similar object ID (also called key) and generates ID using 160 bit SHA.1 function [24]. The object index is responsible for the node ID nodes closest to the object ID (the proximity is measured in a foreign or distance). In the Kademlia model, all nodes are treated as leaves of a two forked trees, and the location of each node is uniquely determined by the shortest prefix of its ID value. For any node, this binary tree can be decomposed into a series of continuous subtrees without its own subtrees. The Kademlia model ensures that each node knows at least one node of its subtrees, as long as these subtrees are not empty. Thus any node can be positioned on the node by the ID value of other nodes.

The Kademlia model uses XOR operation as a measure of the distance between two nodes, assuming that there are two nodes in the Kademlia model, and the node ID is x and y, respectively. Then the distance between them is $d(x, y) = x \oplus y$, such as $x = 1011$, $y = 0010$, $d(x, y) = 1011 \oplus 0010 = 1001$. Although the difference or distance is non-European (non-Euclidean geometric measure), it is reasonable. It is obvious that $d(x, x) = 0$; $d(x, Y) > 0$, if $x \neq y$, and for any x, y, there is $d(x, y) = d(y, x)$, which is called symmetry. In addition, the XOR distance also has a triangle attribute $d(x, y) + d(y, z) \geq d(x, z)$ and triangular attributes from the following two lemmas:

$$d(x, z) = d(x, y) \oplus d(y, z); \quad (1)$$

For arbitrary $a \geq 0$,

$$b \geq 0, \quad (2)$$

there is $a + b \geq a \oplus b$.

The XOR distance is unidirectional in Kademlia models. Unidirectionally refers to any point X and distance $d > 0$, and only the only point y in the network satisfies the $d(x, y) = d$. Unidirectionally guarantees that all positioning of the same data object will eventually get together in the same path, and the more likely the more backward converging will be.

The following is the schematic diagram of the communication implementation for the PaUDT protocol of the P2P model.

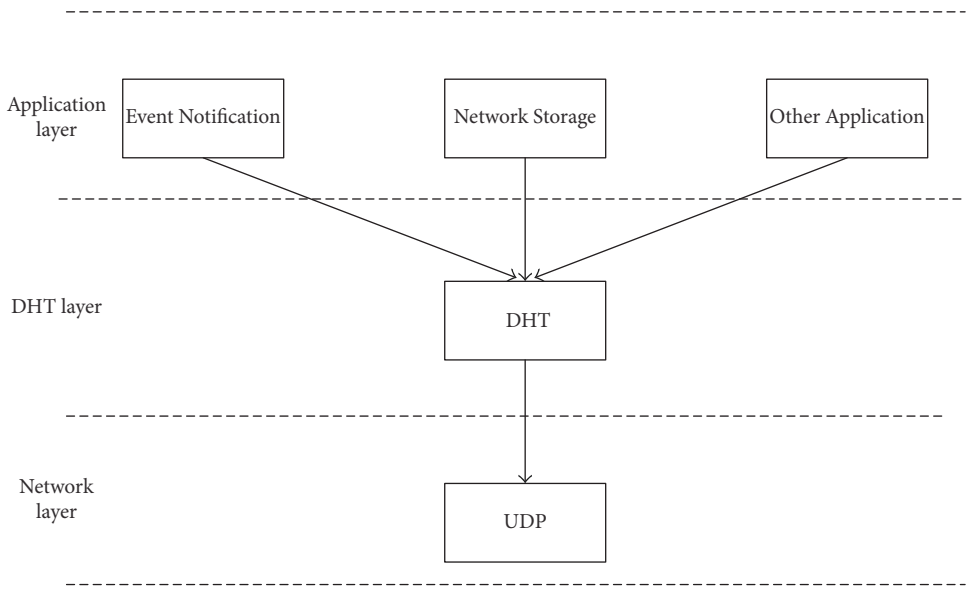

Figure 3: Structure of DHT.

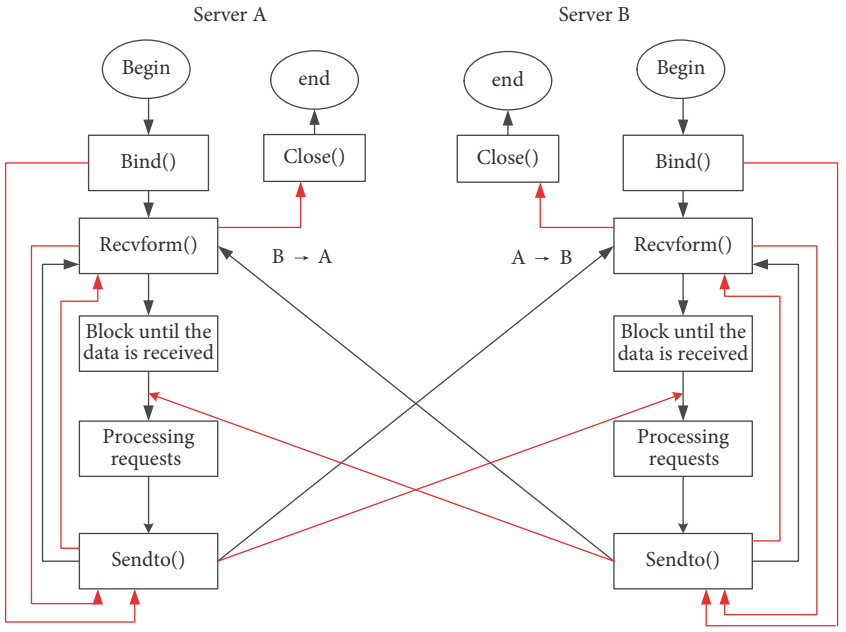

Figure 4: PaUDT communication in P2P mode.

We assume that there are two equal communication sides: server A and server B, and the black line part flow chart is the running process as server; part of the red line flow chart is the running process as the client.

4. RTMFPs Congestion Control Algorithm

In this section, we present the experimental results, which evaluate and characterize the RTMFP and PaUDT, in terms of congestion control algorithm.

RTMFP and UDT are end-to-end, connection-oriented, high reliability, and full-duplex protocol, and there are some same places. For example, they establish connections through four-way handshake, and both application layer protocols which are built on UDP in transport layer. Firstly, the type of data which is transported by RTMFP and UDT is different. RTMFP is used for the real-time and bulk data, but UDT only transports bulk data in the present circumstances. Secondly, unlike UDT, RTMFP is both supporting CS model and P2P model. The last and the major difference is the congestion control algorithm. Through the CCC between RTMFP and UDT, we know which protocol has higher efficiency. Section 2 introduces UDTs CCC simply; this section gives some information about RTMFPs CCC.

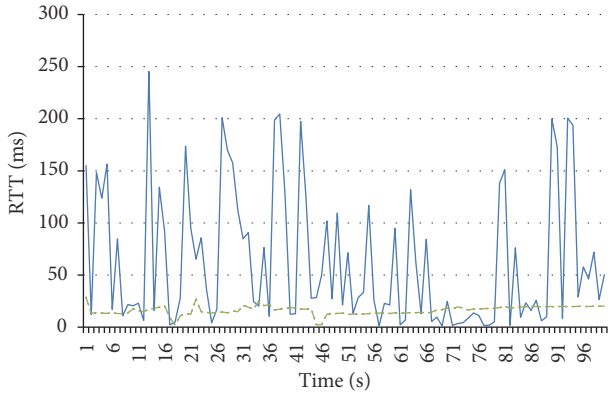

Figure 5: The value of RTMFPs RTT.

A sender of RTMFP increases and decreases its CWnd according to ACK and loss (NAK or timeout) and according to the congestion control and avoidance algorithms [25] like the CCC of UDT. In RTMFP, algorithm also starts from slow start; but not like UDT, slow start's exponential increase rate is adjusted to double every approximately 3 round trips and the multiplicative decrease cuts CWnd by one-eighth on loss and the additive increase is done at half the normal rate (incrementing at 384 bytes per round trip). Because of real-time data, RTMFP has a signal (Time Critical Reverse Notification) which is used to notice other senders who just send original data to decrease the value of CWnd, and if a sender receives a Time Critical Reverse Notification when it is in slow start phase, the sender must stop exponential increase within 800 milliseconds, unless the sender is itself currently sending time critical data to the far end.

From Sections 2 and 3, we know the difference between RTMFP and UDT in congestion control algorithm, and in order to compare which algorithm has more advantages, we design an experiment as follows.

5. Experiments of RTMFP and UDT

In order to contrast which protocol has higher efficiency, we design a experiment in a local area network. This experiment needs six computers; in three computers Linux system for UDT was installed and in others Win7 system for RTMFP is installed.

Install an open platform Adobe Flash Media Server (FMS). Then a computer as server uses FMS to transfer a video to clients, and clients receive the video; at the same time, one client use a capture tool (Wireshark) to capture packets.

After getting these packets, we use a filter tool in Wireshark to choose related packets (these packets are transferred by server), and calculate RTT according to the timestamp of packet's header, through an open UDT source code to transfer the same video with RTMFP. As was stated above, we use Wireshark to capture related packets and calculate UDTs RTT. Contrast the RTT's value of RTMFP and UDT.

As shown in Figure 5, the solid line represents RTMFP's RTT and the change of UDT's RTT is drawn by dotted line. The lines vary greatly. Since a lot of peaks were caused by the curve of RTMFP's RTT, the line of UDT's RTT remains relatively smooth and with smaller value. Although the experiments are in the same environment, the process of receiving video is different. When disposing video, the method of RTMFP is more trouble than UDT. RTMFP plays the video when the video is downloaded, but UDT adopts the method that after this video is received completely, this video is played. Because of different processes, it does not pay special attention to the difference of the value of RTT between UDT and RTMFP, and we need to consider the steadiness of UDT and RTMFP in the experiment.

5.1. Analysis of the Impact of PaUDTs RTT. Based on the system model, performance evaluation is mainly to solve the following problems:

(1) What conditions that network traffic need to satisfy to achieve optimal system performance.

(2) How different parameters influence optimal performance and we set two scenes to analysis the problem.

5.1.1. Calculation of the Most Suitable Maximum Segment Size (MSS). When UDT sends data packets, UDT always tries to pack application data into fixed size packets (the fixed size of data packets is the MSS that negotiated by sender and receiver), unless there is not enough data to be sent [8]. As well known, the bigger the MSS, the smaller the number of packets, but as the buffer of router (MTU) and the error rate in network are limited, the MSS is limited. If the MSS we set in advance is bigger than the value of MTU, this packet will be divided into several fragmentations in network layer, and the receiver must spend more time to reassemble these unordered fragmentations by sequence number, and because in network layer, there is no acknowledge mechanism, if one fragmentation is lost, the sender has to retransfer the whole packet. We know the bigger packets can cause higher error rate and have the danger of segmentation collapse [26], but smaller packets cause more overhead because of their packet headers. So in order to make the total transmission time the shortest and the channel utilization the highest, we must calculate the optimal MSS in the network.

Before calculating the optimal MSS, we give some definitions. P_b is the bit error ratio, P_f represents the wrong frame rate, and l_f is the total length of data part plus control information part and it represents the length of frame and we set its length to N bits. R is the sending rate, and $t_f = l_f/R$ represents the time that a frame is delivered [8, 27]. It is known that the bit error ratio has a relation of the wrong frame rate, and in order to extrapolate the frame error ratio from the bit error rate, CCITT (International Telephone and Telegraph Consultative Committee) puts forward a simple Poisson distribution model or binomial distribution model that is used to describe the random error. Taking the binomial distribution model, we can get the rate ($P_N(m)$) that m bits are wrong in one frame and the result is shown in formula (1) [8, 27]:

$$P_N(m) = C_N^m P_b^m \left(1 - P_b\right)^{N-m}. \quad (3)$$

If one bit in the frame is wrong, this frame is wrong, so we can get the wrong frame rate according to formula (2) as follows:

$$P_f = 1 - P_N(0) = 1 - \left(1 - P_b^N\right). \quad (4)$$

If $P_b \to 0$, according to the Poisson distribution,

$$P_N(m) = C_N^m \times P_b^m \times \frac{(1-P_b)^{N-m}(NP_b)^m}{m!e^{-NP_b}}. \quad (5)$$

So

$$P_f = 1 - P_N(0) = 1 - e^{-NP_b}. \quad (6)$$

When $NP_b \ll 1$, we can get an approximate value of P_f as formula (3):

$$P_f \approx N \times P_b = l_f P_b. \quad (7)$$

We need to calculate the average time of a frame transmission (t_{av}). So as to get an average delay (Delay), we send data in which length is equal to $(n + H)$, so $l_f = n + H = N$. $t_a = A/R$ represents the time that the confirm frame (ACK or NAK) is sent and A is the length of confirmed frame. t_{out} is the minimum time that the retransmission timer is set. In order to get the value of t_{out}, we must know the process time in a node (t_{pr}) and the propagation delay (t_p). The minimum retransmission timer must satisfy a frame that is received by receiver and a sender receives a confirmed frame from a receiver, so $t_{\text{out}} = t_p + t_{\text{pr}} + t_a + t_p + t_{\text{pr}} = 2 \times t_p + 2t_{\text{pr}} + t_a$. t_T is the minimum interval time for the two frames to be successfully sent.

$$t_T = t_f + t_{\text{out}}. \quad (8)$$

So according to the minimal interval time and the wrong frame rate, we can calculate the average time:

$$t_{\text{av}} = \frac{t_T}{\left(1 - P_f\right)}. \quad (9)$$

Now, if the length of file that we want to transport is sum, the average transmission delay equals the average transmission time of a frame multiply by the number of frames which are transported:

$$\text{Delay} = t_{\text{av}} \times \frac{\text{sum}}{n} \quad (10)$$

$$= \frac{(\text{sum}/R)\left[n + H + A + 2\left(t_{\text{pr}} + t_p\right) \times R\right]}{n\left(1 - nP_b - H \times P_b\right)} \quad (11)$$

$$= \frac{(\text{sum}/R)\left(n + H + t_{\text{out}} \times R\right)}{n\left(1 - nP_b - HP_b\right)}. \quad (12)$$

In order to acquire the optimal MSS, we need derivation of Delay:

$$P_b \times n^2 + 2P_b \times \left(H + t_{\text{out}}R\right) \times n - \left(H + t_{\text{out}} \times R\right) \times \left(1 - HP_b\right) = 0. \quad (13)$$

That is,

$$n = \frac{\left(\left(1 - HP_b\right)/P_b\right)}{1 + \sqrt{\left(1 + \left(1 - HP_b\right)\right)/\left(P_b\left(H + t_{\text{out}} \times R\right)\right)}}. \quad (14)$$

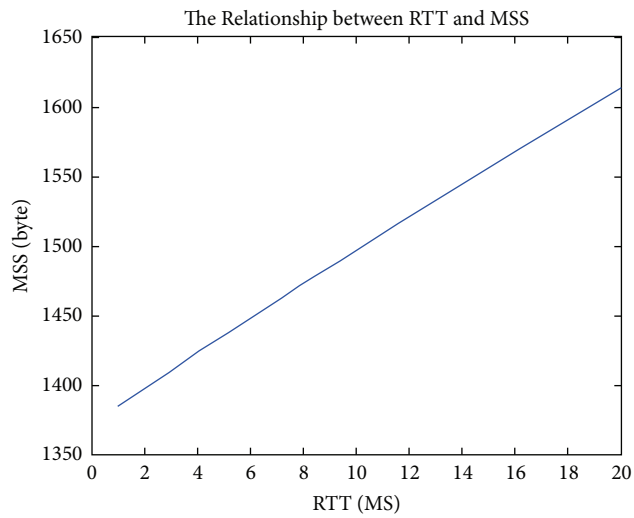

Figure 6: The relationship between RTT and MSS.

If we set A equals H, so we can get a simple new formula (11) from formula (10):

$$n \approx \sqrt{\frac{C}{P_b}} = \sqrt{\frac{2 \times H + 2 \times \left(t_p + t_{\text{pr}}\right)R}{P_b}}. \quad (15)$$

And formula (11) must satisfy the condition that $l_f \times P_b \ll 1$. Figure 6 introduces the relationship between RTT and MSS. Figure 7 describes the relationship between the bit error ratio and MSS. From these two figures, we know that the MSS is advanced with the growth of RTT but is reduced with the growth of the bit error ratio. In the physics experiment, we can get the length of header from wireshark, so $H = 52$ bytes, RTT = 0.01 s, and according to experiments, the average rate in channel is 2 MB/S, so $R \approx 2$ MB/s. $P_b = 6.915 \times 10^{-6}$ bit. So in order to get Figures 6 and 7, we set the rate as 2 MB/S, and when analyzing the effect of RTT on MSS, the bit error ratio is 6.915×10^{-6} bit, and, conversely, when analysing the effect of the bit error ratio on MSS, the RTT is set to 0.01 s.

As shown in Figure 6, there are some values of MSS bigger than 1500 bytes, but now in data link, the biggest value of MSS is 1500 bytes, so we set the value which is equal to 1500 bytes or bigger than 1500 bytes to 1500 bytes.

According to formula (11) and the length of header, the rate in channel, the error bit rate, and the value of RTT, we can get that the optimal MSS equals 1500 bytes.

Although we can get a minimum transmission time using the optimal MSS in network in theory, there may be a practical problem that the MTU is less than the optimal MSS. If the optimal MSS is more than MTU, the packet (the size equals to the optimal MSS) must be fragmented in network layer. So in order to avoid fragment in network, we must set the most suitable MSS in upper layer. This asks us to compare the optimal MSS and the MTU and choose the minimum value as the most suitable MSS.

In order to complete this comparison, UDT is shown in Figure 8. UDT's sender and receiver negotiate the MSS during connection setup. As shown in Figure 8, at first, sender sends

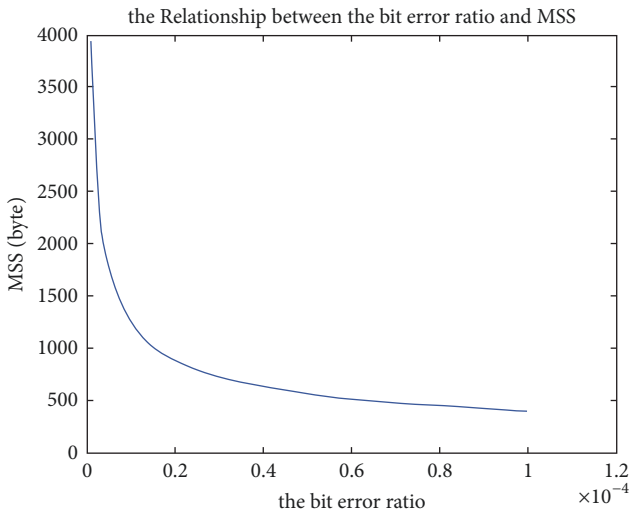

Figure 7: The relationship between MSS and the bit error ratio.

$$\text{MSS} = \sqrt{\frac{2 \times H + 2 \times (t_p + t_{\text{pr}}) \times R}{P_b}};$$

if $rt \rightarrow rt_r mx \cdot rmx_m tu$ **then**
  if $rt \rightarrow rt_r mx \cdot rmx_m tu <$ MSS **then**
    MSS $= rt \rightarrow rt_r mx \cdot rmx_m tuCH$;
  end if
end if

Algorithm 1

a Connection Handshake control packet which includes MSS information; if sender sends packets via a complex network, the MSS will be compared to the MTU of intermediate equipment in network, and if MTU < MSS, the most suitable MSS must equal the minimum MTU, but if MTU > MSS, the most suitable MSS also equals the MSS. After the receiver receives the handshake control packet and gets the most suitable MSS, the receiver sends an ack packet with the most suitable MSS. The sender gets the most suitable MSS and chooses it as the fixed number of a packet. This way has two benefits: first, choosing the minimum MSS as the final MSS can avoid frame being fragmented in network layer and, second, choosing the optimal MSS or the minimum MSS as the final MSS can increase utilization ratio of bandwidth.

In order to achieve the negotiation about MSS, we need to get the value of MTU in intermediate equipment. It is known that there are some structures in the head file of router, for example, reentry and rt_{metrics} structure. In the structure of rt_{metrics}, there is a variable-rmx_{mtu} that is used to get the value of MTU of router and reentry contains the rt_{metrics} structure. In UDT, we set the most suitable MSS as shown in Algorithm 1.

In the statements, $rt \rightarrow rt_{rmx} \cdot rmx_{mtu}$ represents the value of MTU in network, and H is the length of header of UDT.

In our experiments, the MTU of the routers in the data link is 1500 bytes, and the optimal MSS also equals 1500 bytes.

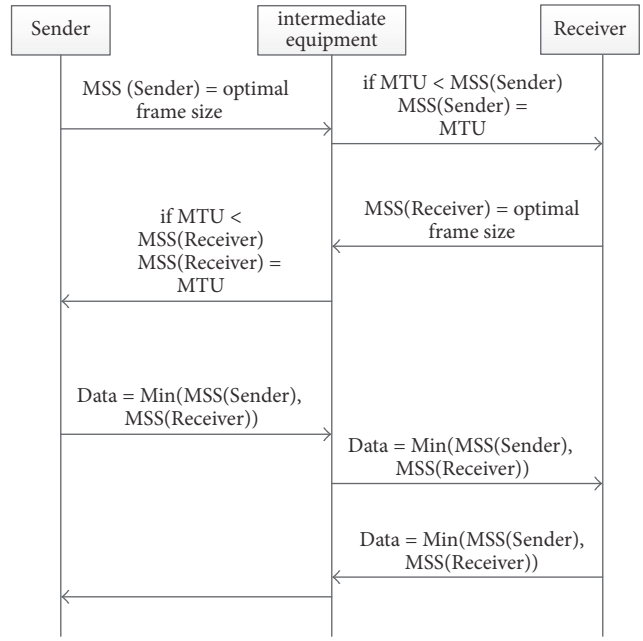

Figure 8: UDT negotiates the MSS between sender and receiver.

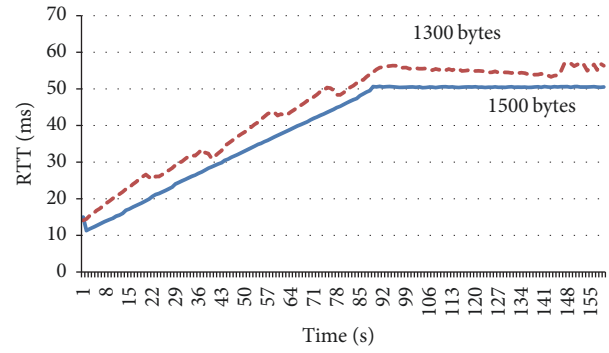

Figure 9: Relationship between UDTs RTT and packet size.

According to the information above, we can get that the real data packet size is 1500 bytes

5.1.2. Verify the Most Suitable MSS and Test the Influence of Different MSS to PaUDTs RTT. In order to check whether the packet size does affect the UDT's RTT and verify whether 1500 bytes is the most suitable MSS in our experiment's environment, we design experiments as follows. In this experiment, we need two computers in a local area network, and every computer is both the sender and the receiver. In order to distinguish them, we call one computer A and the other computer B. Before the experiment, we need to modify the packet size in the open code. In the experiment, we set the packet size to 1500 bytes, 1300 bytes, and 512 bytes, respectively.

When application begins to send data, we record the value of RTT in computer A and computer B. After we get these values, we draw two diagrams like Figures 9 and 10. In the figures, the dotted line represents the value of RTT when the

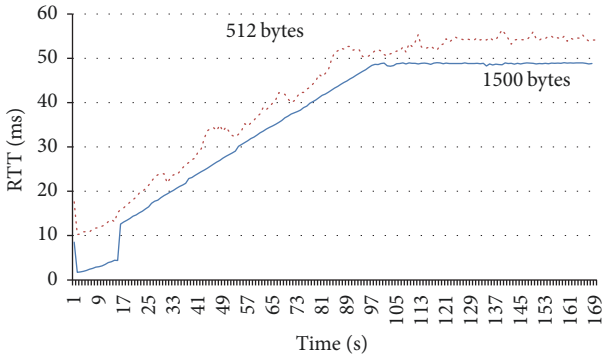

Figure 10: Relationship between PaUDTs RTT and packet size.

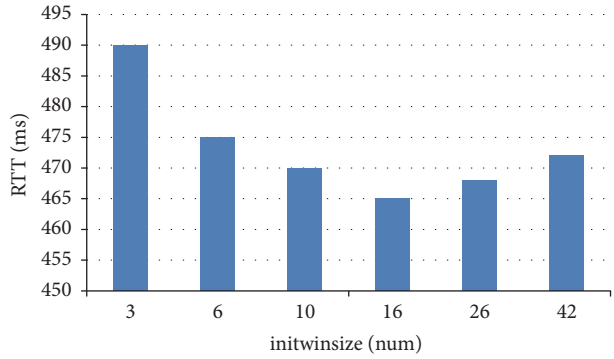

Figure 11: Relationship between initwinsize and latency.

MSS is 1300 bytes or 512 bytes and the solid line represents the RTT which is the result of MSS equaling 1500 bytes.

As shown in Figure 9, when MSS equals the most suitable MSS, the value of RTT is almost increasing linearly at the beginning, and when the time reaches 90, RTT's value stops increasing and the value keeps in range of 50 ms. Conversely, when MSS is equal to 1300 bytes, the value of RTT is not linearly increased. We can clearly see that there are some waves in the growth process. When the value of RTT stops increasing and tends to be stable, the line is not like a solid line. A relatively stable value is maintained, and the value of RTT is higher than 1500 bytes. Figure 10 shows the comparison curve between 1500 and 512 bytes. From the figure, we know obviously that the curve of 1500 bytes is more stable than the curve of 512 bytes and when MSS equals 1500 bytes, the average RTT is smaller than the MSS which equals 512 bytes.

So we can get a conclusion that the different MSS has a relationship to RTT, and when the MSS equals the most suitable MSS, the average RTT is the smallest and the efficiency of packets which are sent or received is the highest.

5.1.3. The Influence of the Size of the Initialize Windows to PaUDTs RTT. There are two variables in PaUDT's congestion control algorithm; one is MSS and the other is initwinsize. Section 5.1 introduces the influence of MSS to PaUDT's RTT, and this section will give some experiments to examine the relationship between initialize windows and PaUDT's RTT.

Before verifying the influence of initwinsize to PaUDT, we introduce the influence to TCP.

Higher initwinsize will offer an advantage for TCP which is reduced latency, as described in formula (12):

$$\left\lceil \log_{\gamma} \left(\frac{S(\gamma - 1)}{\text{initwinsize}} + 1 \right) \right\rceil \times \text{RTT} + \frac{S}{C}. \quad (16)$$

In formula (12), S represents the transfer size, C is bottleneck link-rate, γ is 1.5 or 2 depending on whether acknowledgments are delayed or not, and $S/\text{initwinsize} \geq 1$ [28]. As shown in formula (12), if we increase the initwinsize, the latency will be increased. Although the larger initwinsize can reduce latency, the oversize initwinsize may cause buffer overrun and packet drop, routers experiencing congestion, or congestion collapse.

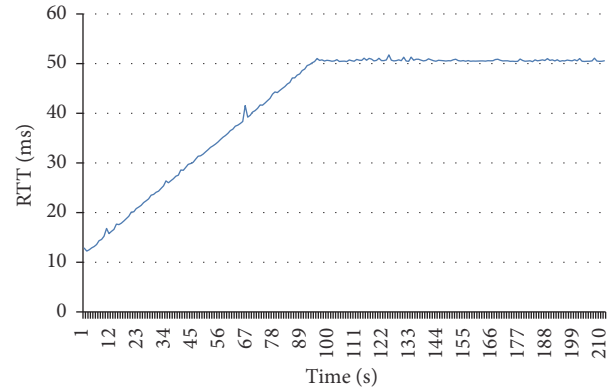

Figure 12: Relationship between PaUDT RTT and initialize window size (8).

Figure 11 shows the average TCP latency for Google Web search as initwinsize is varied from 3 to 42 segments, in a small scale experiment conducted concurrently on six servers in the same data center.

Google Web study shows the effect of initWinseize on TCPs latency and from Figure 11 we know that if the value of initwinsize is 16, the latency is minimum. Section 1 introduces that the congestion control algorithm of TCP begins from slow start algorithm like PaUDT, so the initialize window size also has influence on PaUDT. So we set the default value of initwinsize as 16 depending on the MSS in PaUDT.

In order to confirm the relationship between initwinsize and latency, we set experiments as follows. First, we modify the initwinsize to 8, 16, 32, and 64, respectively.

From Figures 12–14, we can see the difference of RTT as the change of initwinsize. As shown in Figures 12, 13, and 14, the distinction of RTT in different initialize window size is inconspicuous. The RTT value increased from 10 ms to 50 ms and remained stable. These three figures show that when the initwinsize is set to 16, the average of RTT's value is minimum. From the four figures, the most obvious change is Figure 15. When the initwinsize equals 64, the curve which describes the value of RTT causes a lot of volatility.

By analyzing the reasons for the dramatic changes in the RTT value, we found that the changes in intwize affect the value of CWnd. When the sender receives an ACK packet, the value of CWnd is calculated by formula (1). In conclusion,

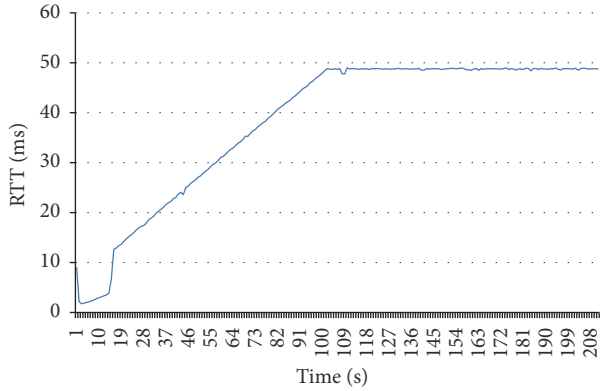

Figure 13: Relationship between PaUDT RTT and initialize window size (16).

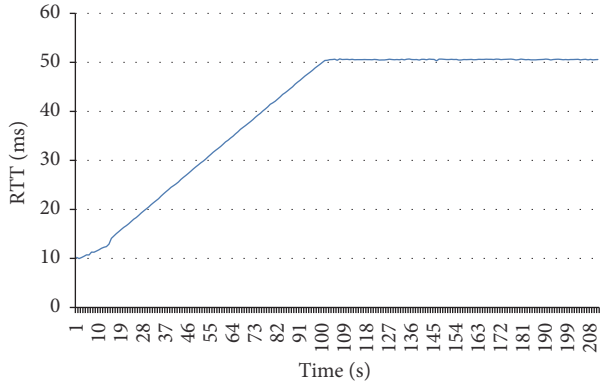

Figure 14: Relationship between PaUDT RTT and initialize window size (32).

initwinsize is bigger, and the value of CWnd is greater, and the excessive CMND may lead to new network congestion.

From Figures 12, 13, 14, and 15, we confirm that the initwinsize also influences the PaUDT like TCP.

6. Conclusion

Fog computing concentrates data, data processing, and applications on devices at the edge of the network instead of storing them almost entirely in the cloud, as does cloud computing. With the expansion of the network scale, the traditional C/S model will increase the burden on the central server, thereby reducing the data transmission efficiency. In this paper, We propose an improved PaUDT protocol with combining the PaUDT and P2P networks. We discussed a number of factors in congestion control algorithm which influence the RTT of PaUDT, applied in IoT-Cloud. In the future, we continue to research the congestion control algorithm of PaUDT and analyze the most important role that parameters play which influences RTT of PaUDT. C/S model increases the load on the server and increases RTT.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

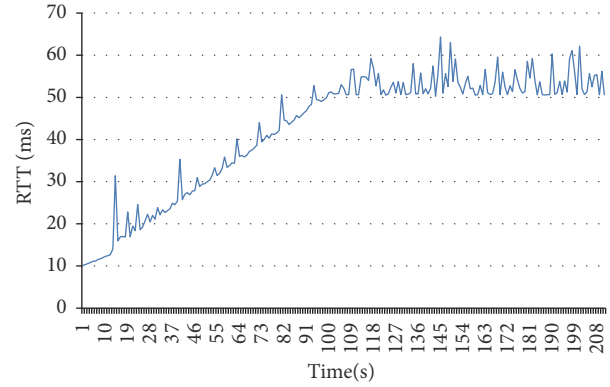

Figure 15: Relationship between PaUDT RTT and initialize window size (64).

Acknowledgments

This work is supported by National Natural Science Foundation of China (6167220961701170), China Postdoctoral Science Foundation funded project (2014M560439), Jiangsu Planned Projects for Postdoctoral Research Funds (1302084B), Scientific & Technological Support Project of Jiangsu Province (BE2016185), and Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks Foundation (no. WSNLBKF201701)

References

[1] I. Lee and K. Lee, "The Internet of Things (IoT): applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] F. Xiao, Z. Wang, N. Ye, R. Wang, and X.-Y. Li, "One more tag enables fine-grained RFID localization and tracking," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 161–174, 2018.

[3] H. Zhu, F. Xiao, L. Sun, R. Wang, and P. Yang, "R-TTWD: robust device-free through-the-wall detection of moving human with WiFi," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1090–1103, 2017.

[4] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the Workshop on Mobile Big Data (Mobidata '15)*, pp. 37–42, ACM, Hangzhou, China, June 2015.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the 1st ACM Mobile Cloud Computing Workshop (MCC '12)*, pp. 13–16, August 2012.

[6] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review Archive*, vol. 44, no. 5, pp. 27–32, 2014.

[7] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, "A roadmap for transmission control protocol (TCP) specification documents," RFC Editor RFC7414, 2015.

[8] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[9] K. G. Jijimol and S. Renjith, "Load-balanced optimal client-server assignment for internet distributed systems," *International Journal of Applied Engineering Research*, vol. 10, no. 69, pp. 216–221, 2015.

[10] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *Computer Communication Review*, vol. 45, no. 5, pp. 123–137, 2015.

[11] E. Kurdoglu, Y. Liu, and Y. Wang, "Dealing with user heterogeneity in P2P multi-party video conferencing: layered distribution versus partitioned simulcast," *IEEE Transactions on Multimedia*, vol. 18, no. 1, pp. 90–101, 2016.

[12] O. Babaoglu, M. Marzolla, and M. Tamburini, "Design and implementation of a P2P cloud system," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*, pp. 412–417, March 2012.

[13] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: a taxonomy, survey and future directions," in *Internet of Things*, pp. 103–130, Springer Singapore, 2018.

[14] F. Xiao, W. Liu, Z. Li, L. Chen, and R. Wang, "Noise-tolerant wireless sensor networks localization via multi-norms regularized matrix completion," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 2409–2419, 2018.

[15] J. Park, H. Jang, and G. Cho, "Congestion degree based available bandwidth estimation method for enhancement of UDT fairness," *Journal of the Institute of Electronics and Information Engineers*, vol. 52, no. 7, pp. 63–73, 2015.

[16] J.-S. Park, D.-S. An, and G.-H. Cho, "Available bandwidth estimation method adaptive to network traffic load considering fairness with UDT flows," in *Proceedings of the 5th International Conference on IT Convergence and Security (ICITCS '15)*, August 2015.

[17] K. T. Murata, P. Pavarangkoon, K. Yamamoto et al., "Multiple streams of UDT and HpFP protocols for high-bandwidth remote storage system in long fat network," in *Proceedings of the 7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON '16)*, pp. 1–6, October 2016.

[18] P. G. Lopez, A. Montresor, D. Epema et al., "Edge-centric computing: vision and challenges," *Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[19] F. Xiao, J. Chen, X. Xie, L. Gui, L. Sun, and R. Wang, "SEARE: a system for exercise activity recognition and quality evaluation based on green sensing," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–10, 2018.

[20] D. V. Bernardo and D. B. Hoang, "End-to-end security methods for UDT data transmissions," *International Conference*, vol. 6485, pp. 383–393, 2010.

[21] W. Feng and P. Tinnakornsrisuphap, "The failure of TCP in high-performance computational grids," in *Proceedings of the ACM/IEEE SC 2000 Conference*, IEEE, Dallas, TX, USA, November 2000.

[22] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, 1997.

[23] N. Shah, A. Ahmad, B. Nazir, and D. Qian, "A cross-layer approach for partition detection at overlay layer for structured P2P in MANETs," *Peer-to-Peer Networking and Applications*, vol. 9, no. 2, pp. 356–371, 2016.

[24] P. Maymounkov and D. Mazières, "Kademlia: a peer-to-peer information system based on the XOR metric," in *Peer-to-Peer Systems*, vol. 2429 of *Lecture Notes in Computer Science*, pp. 53–65, Springer-Verlag, 2002.

[25] M. Thornburgh, "Adobe's Secure Real-Time Media Flow Protocol," RFC Editor RFC7016, 2013.

[26] P. N. D. Bukh, "The art of computer systems performance analysis, techniques for experimental design, measurement," *Simulation and Modeling*, 1992.

[27] Y. Gu, X. Hong, and R. L. Grossman, "Experiences in design and implementation of a high performance transport protocol," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, pp. 6–12, November 2004.

[28] N. Dukkipati, T. Refice, Y. Cheng et al., "An argument for increasing TCP's initial congestion window," *Computer Communication Review*, vol. 40, no. 3, pp. 26–33, 2010.