Received January 18, 2020, accepted February 8, 2020, date of publication February 17, 2020, date of current version February 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2974191

Multipath Routing and MPTCP-Based Data Delivery Over Manets

TONGGUANG ZHANG[1], SHUAI ZHAO[2], AND BO CHENG[2], (Member, IEEE)
[1]School of 3D Printing, Xinxiang University, Xinxiang 453003, China
[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Tongguang Zhang (jsjoscpu@163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003804, in part by the Key Scientific and Technological Projects in Henan Province under Grant 202102210146, in part by the National Natural Science Foundation of China under Grant 61921003, in part by the Beijing Natural Science Foundation under Grant 4182042, and in part by the Beijing Nova Program of Science and Technology under Grant Z191100001119031.

ABSTRACT In some special circumstances (e.g., tsunamis, battlefields, and earthquakes), communication infrastructures are damaged or nonexistent. For communication among people, mobile smart devices (MSDs) can be used to construct mobile ad hoc networks (MANETs). This paper focuses on the problem of data delivery in MANETs aiming to improve the quality of service (QoS) and quality of experience (QoE) users receive. MANETs, however, have the well-known problems of frequent disconnections and high rates of failed transmissions as MSDs move in and out of network coverage areas, and the topology constantly changes. To solve these issues, the main contributions of this work are as follows: (1) we provide and investigate the QoE-driven multipath TCP (MPTCP)-based data delivery model in MANETs; (2) we present hidden Markov model-based optimal-start multipath routing, which can effectively predict a mobile node's near future network connection state according to its past connection state; (3) we leverage MPTCP to simultaneously transmit data via multiple interfaces of MSDs and improve the establishment method for MPTCP subpaths; and (4) we study and improve the algorithm of multihop routing in MANETs. The test results show that our algorithms can offer more efficient use of multiple subpaths and better network traffic load balancing than using standard MPTCP alone.

INDEX TERMS Data delivery, hidden Markov, MANET, multipath routing, multipath TCP, quality of experience.

I. INTRODUCTION

Natural disasters, e.g., earthquakes, tsunamis, and battlefields, often cause breakdowns or interruptions in communication infrastructures. An example is the Wenchuan earthquake, which had a magnitude of 8.0 Ms/7.9 Mw and occurred at 14:28:01 China Standard Time on May 12, 2008. Over 69,000 people lost their lives in the quake. In the aftermath of earthquakes, it is vital to speed up earthquake disaster relief. The rescue center must determine what resources are needed to shelter people and to help people recover from the disaster. Hence, it is essential for rescue centers to identify the location of disaster victims who may be trapped or isolated and to bring them to safety and medical attention. To this end, it is important to provide concise position information. However, fast, reliable damage assessment is extremely difficult, particularly in large disaster earthquakes. To evaluate the scope and severity of earthquakes and improve the efficiency of search-and-rescue, advanced damage-detection methods are needed. One key to enhancing the effectiveness of response is to adopt communication technologies. It is necessary to provide normal communication among people or mobile smart devices (MSDs). Smartphones are more powerful and can be used to construct mobile ad hoc networks (MANETs). The appeal of MANETs is the capacity for rapid deployment and reorganization. In MANETs, each node can serve as a router and host. Basic types of routing algorithms are single-hop and multihop. In multihop MANETs, nodes cooperate to relay traffic on behalf of one another to reach remote nodes. Mobile applications are sensitive to latency, as the quality of experience (QoE) is negatively affected when data are delayed. Due to the mobility of MSDs and the multihop wireless transmission, MANETs lack the performance and reliability of wired networks, which results in a reduced

The associate editor coordinating the review of this manuscript and approving it for publication was Quansheng Guan.

quality of service (QoS). Currently, it is a challenge to provide reliable data delivery over MANETs in disaster areas.

Now, MSDs are equipped with multiple communication interfaces (e.g., WiFi, 4G/5G, etc.). Various methods for improving the QoE of latency-sensitive mobile applications are active subjects of research [1]. However, a weakly explored area is determining whether utilizing all available network interfaces on the MSDs could improve the QoE. Thus, it is necessary to optimize the QoS of mobile applications and improve the QoE. For instance, how to efficiently use multiple network interfaces simultaneously to reliably transfer data for fewer failed communications. Multipath transmission is an ideal solution. Unfortunately, the current dominant transport layer protocols (e.g., TCP) cannot control multiple paths. The main problem with the existing protocols is that mobile applications cannot change a communication session to another path. Hence, we use multipath TCP (MPTCP) [2]–[5], a new transport layer protocol, to solve this issue. MPTCP is a TCP extension and is being standardized by the Internet Engineering Task Force (IETF). MPTCP provides concurrent data transmission over multiple paths and supports compatibility with a single TCP. In this case, there is only one network interface, MPTCP creates only one subflow on the interface and requires the stack on both ends. MPTCP can help to improve the utilization of network resources by aggregating multiple communication interfaces.

Although MPTCP has been shown to be more resilient to link failures and can aggregate capacity to provide more throughput, because it still uses TCP in its subflows, there are some disadvantages of MPTCP with unreliable connections. For example, subflows are built on several links that will be disconnected, rather than using links that will remain connected in the future. Thus far, whether MPTCP is suitable for mobile applications transmitting latency-sensitive traffic in special emergency MANET scenarios has not been thoroughly investigated. The lack of performance and reliability in MANETs is a challenge for reliable and efficient data communication. To achieve reliable and efficient multipath data transmission, we provide and investigate the QoE-driven MPTCP-based data delivery model in MANETs. In the proposed model, we consider both symmetric and asymmetric multipath communication and set up various levels of emergency applications. Each level corresponds to a set of QoS parameters that will be transferred to the network layer and then act on route generation, path selection, packet scheduling and so on. These are transparent to users.

In this paper, we have four main contributions. **First**, to achieve optimal data transmission in MANETs constructed by MSDs with multiple network interfaces, we provide and design a QoE-driven MPTCP-based data delivery model. **Second**, we present the hidden Markov model (HMM)-based optimal-start multipath routing scheme, which can effectively predict a mobile node's near future network connection state according to its past connection state and improve OSPF (Open Shortest Path First) MANET Designated Routers (MDR) to add routing table entries according to the number of next-hops and network interface number. **Third**, we improve the establishment method for MPTCP subpaths. To the best of our knowledge, this is the first time that the method of adding routing table entries and the method of establishing MPTCP subpaths have been simultaneously considered to offer more efficient use of multiple subpaths and better network traffic load balancing to increase throughput and reliability. **Finally**, we implement and evaluate the data delivery model in MANETs.

The rest of the paper is organized as follows. Section II discusses related work. Section III introduces an overview of the proposed QoE-driven MPTCP-based data delivery model in MANETs. Section IV discusses MPTCP in QoE-Oriented MANETs and provides a QoE-driven packet scheduling framework and algorithm. Section V presents the implementation of MPTCP in MANETs and shows the measurement results. Finally, Section VI concludes the paper and discusses possible future work.

II. RELATED WORK

QoE enhancement for real-time data transmission in MANETs is a challenging and important issue. The authors in [6] developed an optimal bandwidth allocation strategy in MANETs. The authors in [7] analyzed major factors influencing the QoE of voice communication in MANETs. However, they did not consider multipath transmission using MPTCP. Multipath provides new opportunities for improving mobile application performance. Hence, industry has also been enthusiastically adopting MPTCP. In September 2013, some built-in apps in iOS, e.g., Siri, began supporting multipath [8]. MPTCP v0.86 was ported to Android 4.4.4 on Samsung Galaxy S3 smartphones [2]. MPTCP v0.89.5 was ported to Android 4.4.4 [9]. By adding redundancy to data segments, it is possible to improve the performance of MPTCP in lossy environments [10], [11].

Paasch *et al.* [12] provided a detailed study of MPTCP schedulers and their impact on performance. To date, several schemes based on MPTCP have been proposed to prevent goodput degradation. In [13], congestion window adaptation and a proactive scheduler were proposed to prevent goodput degradation. Although some transmission strategies have been proposed in terms of delay [12], [14], these schemes mainly focus on the buffer blocking phenomenon. One solution is to schedule data to subflows to independently operate both application-dependent functions and network-dependent functions. In [12], the impact of scheduling in MPTCP was evaluated, and MPTCP was shown to improve performance by scheduling strategies considering network conditions. Some scheduling schemes [14], [15] have been proposed to prevent goodput degradation. The authors in [16] showed how path selection for a subflow affects network throughput. The authors in [12] addressed the problem of head-of-line blocking in the network. In MPTCP, this phenomenon is caused by the packets that are scheduled on the low-delay subflow, which have to wait for the high-delay subflow's packets to arrive in the out-of-order queue of the receiver.

QoE-driven packet scheduling is based on MPTCP and has already been extensively investigated in existing studies [17]–[23], and each work focused on a very specific and detailed problem, e.g., mobile energy consumption, video streaming quality, and throughput optimization. We consider the QoE optimization for only one application in our work because the QoE metrics for different applications are completely different. For instance, delay is the most important for audio applications, while video applications have widely used QoE metrics, such as the structural similarity score (SSIM) [24]–[26] and rebuffering ratios. The authors in [27] explored concepts and feasibility of realizing MPTCP with path awareness, in which the path-aware information was leveraged to reinforce the MPTCP transmissions. The authors in [28] developed a comprehensive approach that is capable of assessing the performance of long-lived MPTCP flows with joint WiFi and cellular network access. Relying on a parallel queueing model, the authors developed a framework that features the controllable network parameters, such as the retransmission limit and the buffer sizes, to capture their impact on the TCP-level performance. However, they did not consider both the method of adding routing table entries and the method of establishing MPTCP subpaths simultaneously. The proposed MPTCP scheduling is modified based on the proposed multipath routing in this paper.

Several difficult problems (e.g., routing, reliability and QoS) can arise due to the frequent disconnections of nodes in MANETs. How to generate a routing table is the core issue. Many routing protocols have been developed for MANETs, which differ in their protocols characteristics, techniques used, and network structure [29]. The two most popular dynamic routing protocols are distance vector routing and link state routing, both of which require each router to periodically broadcast routing advertisements. These routing algorithms are not efficient in MANETs. Thus far, routing protocols used in MANETs can be classified as table-driven, on-demand, and hybrid [30].

III. QoE-DRIVEN DATA DELIVERY MODEL IN MANETS

A. QoE-DRIVEN DATA DELIVERY MODEL

It is essential to deliver real-time data such as audio and video in MANETs. Therefore, MANETs must be able to provide the required QoS for the delivery of real-time data that request a QoS routing and improving user QoE. QoS is defined as a set of measurable prespecified service requirements such as delay, bandwidth, packet loss, and jitter that a network needs to make them available for the end-users while transporting a packet stream from a source to its destination [30]. QoE extends the QoS by making statements about the user experience. In contrast to QoS, QoE does not provide numeric values about the bandwidth. QoE is a user-centric concept reflecting the end-user satisfaction of a service. QoE is identified as the degree of satisfaction or annoyance of the user with an application or service [31]. However, in MANETs, there are some challenges that need to develop efficient routing procedures, packet scheduling for efficient use of partial bandwidth and communication capacity. Thus, to provide efficient QoS and high QoE in MANETs, we provide and design a QoE-driven data delivery model, which contains 3 parts: users, applications and network (TCP/IP) in MANETs, as shown in Fig. 1.

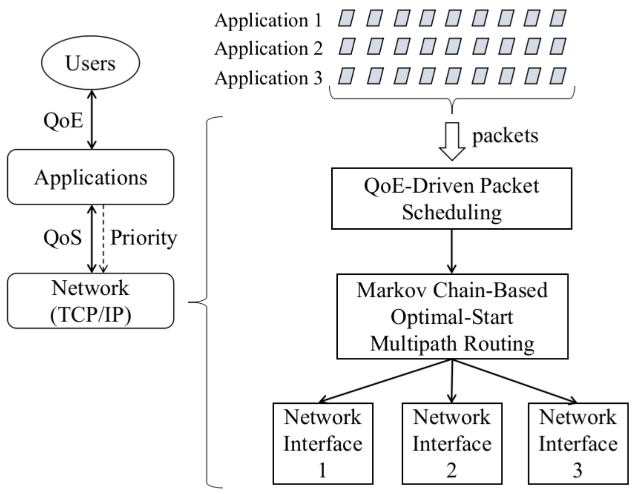


FIGURE 1. QoE-Driven Data Delivery Model in MANETs.

TABLE 1. Description of QoS parameters.

Parameters	Description
bandwidth	the rate at which a request's traffic must be passed by the network
delay	time to deliver a packet to its destination
jitter	the variation in latency
packet loss	the percentage of lost data

Fig. 1 shows the relationships among QoE, QoS, users, applications, and network (TCP/IP). In the proposed model, we simultaneously consider both symmetric and asymmetric multipath communication. Various priorities are set up for different types of applications, and each priority corresponds to a set of QoS parameters. Users do not have to consider QoS and QoE that are transparent to them. Usual QoS metrics include bandwidth, delay, loss rate and jitter. The relationship between the proposed scheduling and proposed multipath routing is that the former is modified based on the latter. A detailed description will be given in Section IV.

The application layer shows the operation method of application, which is directly related to the real experience perceived by users.

The network layer is the key to the model and involves the QoS parameters listed in Table 1. QoS parameters act on route generation, path selection, packet scheduling and so on.

B. APPLICATION PRIORITY IN MANETS

Different applications have different network requirements, which can be described in terms of metrics such as bandwidth, delay, jitter and packet loss. Real-time applications require more bandwidth and are sensitive to latency in MANETs. It is difficult to support different applications with appropriate QoS in MANETs with MPTCP. In some situations, applications with low-priority and high-bandwidth usage can consume so much bandwidth that other high-priority applications become unavailable or limited. This case can lead to a decrease in users' QoE. Hence, it is impossible to meet the needs of various applications simultaneously. In fact, all parameters considered for packet scheduling have already been discussed in previous studies. In our proposed model, different priorities are assigned to different applications. The application priority dictates a set of QoS parameters required from the network for a specific application. The application priority is used to assist in packet scheduling, which attempts to match each defined application to the most appropriate network resource.

C. HMM FOR NODE CONNECTION STATE IN MANETS

Disconnections are quite common in MANETs. To improve the packet delivery ratio and QoS of data transmission, it is necessary to devise efficient routing schemes. However, it is a challenge to design an efficient routing scheme to deliver a packet in a timely manner with a low drop rate in MANETs. Due to the mobility of the node and the periodicity of routing table updating, the current valid routing entries may soon become invalid in the update cycle, and QoS might decease. The key to solving this problem is to effectively predict the future connection state for each node based on its past connection state. The HMM is exploited to make predictions; hence, we devise the HMM-based optimal-start multipath routing scheme to improve the quality of routing and ensure more reliable and timely data transmission in MANETs. Our proposed scheme chooses the node that has a better connection state as the next-hop. This section focuses on HMM for the node connection state in MANETs. In the next section, we detail the HMM-based optimal-start multipath routing scheme for MANETs.

HMM [32] is a statistical Markov model with hidden states and observable states. Each node collects and maintains the transition probability matrix and the state sequence. In an HMM, observable states depend on the hidden states, and each state has a probability distribution over the possible observable states. The only method for evaluating hidden states is to observe the observable states. The transition between hidden states follows the Markov process in which the current state only depends on the previous state.

Fig. 2 illustrates the hidden states and observable states for the HMM in MANETs. The observable states include good moderate and bad, good denotes continuous network connections, moderate denotes intermittent network connections, and bad denotes network disconnections. Observable states can be measured according to packet loss rate, signal strength or RTT (Round-Trip Time).

Table 2 shows the definitions of the observable states. if_quality[i] in **Appendix B** is related to these states. The thresholds for a given metric are set in **Appendix B**. The 'if_quality' denotes the quality of a network interface. The initial (maximum) value of if_quality is 10000. In the running process of the system, the value of if_quality is modified according to whether packet is lost, whether retransmission timer times out, and whether ACK is received. The procedure update_if_quality in **Appendix B** is responsible for adjusting the value of if_quality.

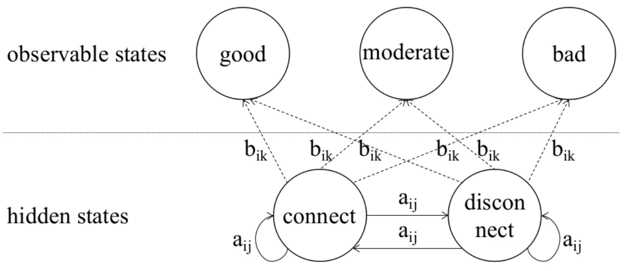


FIGURE 2. Hidden states and observable states for HMM in MANETs.

TABLE 2. State definition.

Observable state	if_quality
good	8001–10000
moderate	5001–8000
bad	0–5000

Viterbi algorithm [33], [34] is used to find the most likely sequence of connection (hidden) states. The algorithm is modeled as a six-tuple HMM $\lambda = (H, O, \pi, Y, A, B)$ where

$H = \{h_1, h_2, \ldots, h_N\}$, the hidden state space, N is the number of hidden states. E.g., H = {connect, disconnect}.

$O = \{o_1, o_2, \ldots, o_M\}$, the observation space, M is the number of observable states. E.g., O = {good, moderate, bad}. Observable states are measured according to packet loss rate, signal strength or RTT (Round-Trip Time).

$\pi = \{\pi_i\}$, $1 \le i \le N$, the initial hidden state probability vector, where $\pi_i = (\pi_1, \pi_2, \ldots, \pi_N) = P(H_i)$, $1 \le i \le N$ and $\sum_{i=1}^{N} \pi_i = 1$.

$Y = \{y_1, y_2, \ldots, y_T\}$, the sequence of observations.

$A = \{a_{ij}\}$, the N×N state transition matrix, the transition probabilities between the hidden states H_i and H_j, where

$a_{ij} = P(q_t = H_j \mid q_{t-1} = H_i)$, $1 \le i, j \le N$, $a_{ij} \ge 0$, ∀i,j and $\sum_{j=1}^{N} a_{ij} = 1$, ∀i.

$B = \{b_{ik}\}$, the N×M state dependent emission matrix, the probabilities of the observable states O_k in the hidden state H_j, where $b_{ik} = P(O_k|H_i)$, $1 \le i \le N$; $1 \le k \le M$ and $\sum_{k=1}^{M} b_{ik} = 1$, ∀i.

The procedures main and hmm_predict give the connection states evaluation process in **Appendix A**. Due to Viterbi algorithm [35], [36] uses dynamic programming, it's time complexity is $O(T^*|N|^2)$ for any HMM with N states and an observation sequence of length T. To analyze the prediction overhead, we carried out the command "*time./viterbi_algorithm 2 1 0*" 20 times on a notebook with 64G mem and an Intel Core i7-8750H processor. The execution times are 0.005s, 0.004s, 0.004s, 0.003s, 0.003s, 0.005s, 0.002s, 0.005s, 0.005s, 0.005s, 0.003s, 0.005s, 0.004s, 0.002s, 0.005s, 0.004s, 0.005s, 0.003s, 0.004s and 0.005s respectively. The source code for the Viterbi algorithm is available from [51]. As shown in **Appendix A**, the procedure hmm_predict is executed every 2 seconds or 1 second. Therefore, the computation overhead will not affect the update of routing table.

D. HMM-BASED OPTIMAL-START MULTIPATH ROUTING

When designing MANETs, several difficult problems can arise due to the frequent disconnections of nodes. How to generate a routing table is the core issue. In this paper, we improve OSPF-MDR [37] to generate routing tables. OSPF for IPv6 is also referred to as OSPF version 3 (OSPFv3). OSPF-MDR is an extension of OSPFv3 [38] to support MANETs and is based on the selection of a subset of MANET routers, consisting of MANET Designated Routers (MDRs) and Backup MDRs (BMDRs). OSPF-MDR uses data in the Hello packets to elect the MDR and BMDR. The MDRs form a connected dominating set (CDS). For robustness, the MDRs and BMDRs together form a biconnected CDS that is constructed using 2-hop neighbor information provided in a Hello protocol extension. It is important to provide guaranteed QoS routing in MANETs. QoS routing needs to mainly manage real-time communication. Multipath routing generally consists of three components: route discovery, route maintenance, and traffic allocation. The function of traffic allocation is implemented by MPTCP, which is discussed in Section IV.B. Route discovery and route maintenance are responsible for multiple routes between source and destination node pairs.

Fig. 3 shows how to maintain the multipath routing table and propagate to all the nodes in MANETs. Both types of Hello (full and differential) are supported in OSPF-MDF. Full and differential Hellos are used for neighbor discovery and for enabling neighbors to learn 2-hop neighbor information. Full Hello packets always include the list of neighbors. Differential Hello packets only include the neighbors whose state has recently changed and are used to reduce control overhead. Therefore, differential Hellos allow OSPF-MDR to react quickly to the dynamic topology of MANETs. OSPF-MDR has a Link-State DataBase (LSDB) composed of Link State Advertisements (LSAs) and synchronized between adjacent nodes. When two neighboring nodes become adjacent, they synchronize their LSDBs. Each node sends the other node a set of Database Description (DD) packets that describes the node's LSDB. This is done by listing the header of each LSA in one of the sent DD packets. Each node can determines whether the other node has newer LSAs that should be requested via Link State Request (LSR) packets. Initial LSDB synchronization is performed through the exchange of DD, LSR, and Link State Update (LSU) packets. Thereafter, LSDB synchronization is maintained via flooding, utilizing LSU and Link State Acknowledgment (LSAck) packets. Since the CDS is responsible for flooding new LSA (its payloads carried by the LSU packets), the flooding procedure in OSPF-MDR is called CDS flooding. As shown in Fig. 3, each node is either in the CDS or one hop away from it.

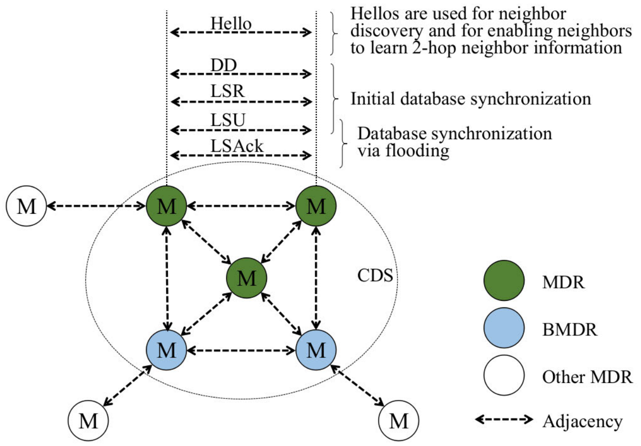

Hello
Hellos are used for neighbor discovery and for enabling neighbors to learn 2-hop neighbor information
DD
LSR
Initial database synchronization
LSU
LSAck
Database synchronization via flooding
M
CDS
MDR
BMDR
Other MDR
Adjacency

FIGURE 3. Maintain the multipath routing table and propagate to all the nodes in MANETs.

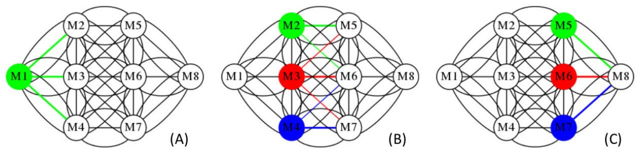


FIGURE 4. Optimal-start multi-interface multipath routing between M1 and M8.

MDRs exchange LSA among each other and relay them its neighbors. Nodes ignore LSA from neighbors with which they do not have a bidirectional connection. The LSDB is pieced together from LSAs. From the LSDB, a routing table is calculated by constructing a shortest-path tree. In MANETs, each node is equipped with multiple network interfaces. It is necessary to use multiple interfaces simultaneously for transmitting data. OSPF-MDR generates routing table entries for one network interface of a node by default. In this paper, we improve OSPF-MDR by generating routing table entries for other network interfaces as soon as for one network interface of a node. Therefore, this improvement can reduce the computing overhead of mobile nodes while generating a complete routing table. Specific improvement methods are detailed in the later part of this section and in Section V.A.1.

Now, we examine the issues of multipath routing in MANETs. Multipath routing is typically proposed to increase the reliability of data transmission (i.e., fault tolerance) or to provide load balancing and higher aggregate bandwidth. Multipath routing allows the establishment of multiple paths between source and destination node pairs. To clearly demonstrate this situation, see Fig. 4, where node M1 establishes three paths to node M8. If node M1 sends the same packet along all three paths, as long as at least one of the paths does not fail, node M8 will receive the packet, which demonstrates how multipath routing can provide fault tolerance. In addition, simultaneously using multiple paths to route data may increase the aggregate bandwidth; therefore, the bandwidth requirement of mobile applications can be satisfied.

In MANETs, the routing entries for each destination contain a list of the next-hops along with the corresponding hop counts. There is a key problem, when a source node sends data along multiple routes, some or all of the routing entries may become obsolete due to node mobility. Therefore, routing protocols need to update the routing table; however, performing a route discovery would result in a delay before new routes are available. This may degrade the QoS of the application. For more robust and reliable communications, it is important that a mobile node can predict the connection state of the next-hop. In these conditions, first, OSPF-MDR is used to periodically find the latest routing entries, and the periodic interval is 2 s. HMM is used to predict the connection state of the next-hop after 2 s and decide whether to add it to the routing table based on the state. For example, in Fig. 4, at time T1, OSPF-MDR on node M1 finds three next-hops to node M8 and uses HMM to predict that the connection states of the three next-hops at time T1+2 s are good; therefore, the three next-hops (node M2, M3 and M4) are added into the routing table on node M1. At time T1+2s, OSPF-MDR on node M1 finds three next-hops to node M8 and uses HMM to predict that the connection states of two next-hops (node M2 and M3) at time T1+4s are good, the connection state of the next-hop (node M4) at time T1+4s is poor; therefore, the two next-hops (node M2 and M3) are added into the routing table on node M1.

Another key problem is that disjoint routes offer certain advantages over nondisjoint routes. In principle, node-disjoint routes offer the most aggregate resources and higher fault tolerance. In node or link-disjoint routes, a link failure will only cause a single route to fail. Node-disjoint routes offer the highest degree of fault tolerance. The main advantage of nondisjoint routes is that they can be more easily discovered. Node-disjoint routes are the least abundant and are the hardest to find. Hence, some researchers want to improve network performance through node or link disjoints. For example, MPTCP mainly relies on open shortest path first/equal cost multipath (OSPF/ECMP) to hash different subflows to different paths. However, the authors in [39] indicated that OSPF/ECMP routing is not flexible if all the subflows of a connection use the same path. To solve the problem, the authors in [40] used k disjoint paths to route the subflows of an MPTCP connection. However, the authors in [41] proved that four versions of the problem (i.e., the graph is directed or undirected and the paths are link-disjoint or node-disjoint) are strongly NP-hard, even if k equals two. It costs exponential time to find an optimal solution for the above problems. Because the problem is NP-complete, we propose an optimal-start multipath routing scheme that does not consider this NP-hard problem of k disjoint paths. However, the problem cannot be solved fundamentally by using only routing; we simultaneously consider using multi-interface multipath routing and multipath transmission technology.

In MANETs, each mobile node is equipped with multiple network interfaces. Simultaneously using multiple interfaces can create many benefits for users. However, the different types of connectivity must be considered. For example, it is necessary to distinguish between WiFi and 4G/5G. First, the weight of each route is set up based on an accurate link state. Then, MPTCP chooses routing according to the weight to route the subflows of an MPTCP connection. We will detail the multipath transmission technology in Section IV.

As shown in Fig. 4, each node has multiple interfaces, and the original OSPF-MDR may obtain all of the available next-hops and add them to the routing table. For example, when node M1 finds next-hops to node M8, it can obtain 9 next-hops; that is, there are 3 next-hops between the pair of nodes M1 M2, M1 M3 and M1 M4. Based on the routing table, MPTCP may use the paths between the same pair of nodes, which leads every subflow to compete for the same resources in the same pair of nodes. We, hence, improve the OSPF-MDR for MANETs. Our proposed algorithm is divided into three phases. The first phase is to find the candidate next-hops. Due to the timeliness of routing items, it is necessary to use HMM-based prediction technology. The second phase is to predict the connection states of the next-hops by using HMM. The third phase is to update the routing table using the next-hops with a good connection state. The greatest advantage of the scheme is that it can reduce the computing overhead of mobile nodes.

Take Fig. 4 as an example to illustrate the results generated by the HMM-based optimal-start multipath routing. In this example, we give all the next-hops from node M1 to node M8.

We define a function **hmm(x1, x2)**, which can find the best next-hop between node x1 and node x2 based on HMM.

As shown in Fig. 4 (A), the next-hops from node M1 to node M8 are hmm(M1, M2), hmm(M1, M3) and hmm(M1, M4).

As shown in Fig. 4 (B), the next-hops from node M2, M3, M4 to node M8 are as follows.

M2: hmm(M2, M5), hmm(M2, M6)

M3: hmm(M3, M5), hmm(M3, M6), hmm(M3, M7)

M4: hmm(M4, M6), hmm(M4, M7)

As shown in Fig. 4 (C), the next-hops from node M5, M6, M7 to node M8 are hmm(M5, M8), hmm(M6, M8), hmm(M7, M8), respectively.

Appendix A gives the HMM-based optimal-start multipath routing algorithm, which is an extension to OSPF-MDR.

E. QoE-DRIVEN MULTIPATH TRANSMISSION

We now consider the situation where HMM-based optimal-start multipath routing is used, and the multiple paths are scheduled simultaneously to transfer data by MPTCP. Fig. 5 (A) shows the subpaths created by the original MPTCP based on the original routing table, which clearly leads every subflow to compete for the same resources in the same pair of nodes. Fig. 5 (B) shows the subpaths created by the improved MPTCP based on the HMM-based optimal-start routing table. MPTCP can use multipath for a single MPTCP connection and hash different subflows to different subpaths. This improves the robustness of the MPTCP connection.

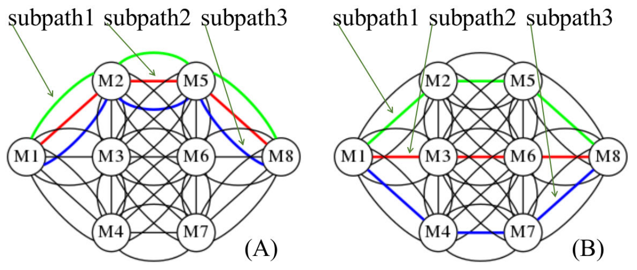


FIGURE 5. Creating subpaths between M1 and M8.

However, there is the problem of head-of-line blocking in MANETs, which is caused by the packets that are scheduled on the low-delay MPTCP subflow that have to wait for the high-delay subflow's packets to arrive in the out-of-order queue of the receiver. In addition, it is important to consider the real-time voice and video communications that may benefit from bandwidth aggregation. Multipath transmission has great promise. However, the optimal allocation of traffic to multiple paths is challenging. Thus, we propose a new scheduling scheme, QoE-driven packet scheduling, which schedules packets to subflows according to network delay and bandwidth, aiming to achieve efficient concurrent transmission over multiple paths. The scheme is detailed in Section IV.B.

F. SCHEME OF USING QoS PARAMETERS

Many applications have specific network requirements (QoS parameters). To improve user QoE in MANETs, the QoS parameter selection should be a black box to users. Therefore, we present a scheme for transferring application priority from user space to kernel space via socket, as shown in Fig. 6. In the scheme, applications for special tasks in MANETs are bound with a priority that is transparent to users. Once an application starts up, the priority of which application is transferred into the kernel and the available network resource can be intelligently allocated.

IV. MPTCP IN QoE-ORIENTED MANETS

MPTCP uses multiple subflows to implement concurrent multipath transport [42]. Each subflow is defined by a source/destination IP address pair and appears as a regular TCP connection. MPTCP can manage many paths, improve the transmission performance and robustness of end-to-end connections, and automatically transfer the traffic from the congested path to the better path. In addition, MPTCP can be backward compatible with traditional TCP. Because Android is a popular mobile operating system, Android Things is an Android-based operating system for MSDs in IoT; thus, we research MPTCP Linux implementation (MPTCP-L) [43] and implement it on MSD with Android installed. As shown in Fig. 6, MPTCP-L consists of several main parts: a meta socket, a master subsock, slave subsocks, a multipath control block (MPCB) [44]), path managers, schedulers, reorders, and congestion controllers [45], [46]. If MPTCP is enabled on MSD with Linux/Android installed, when an application

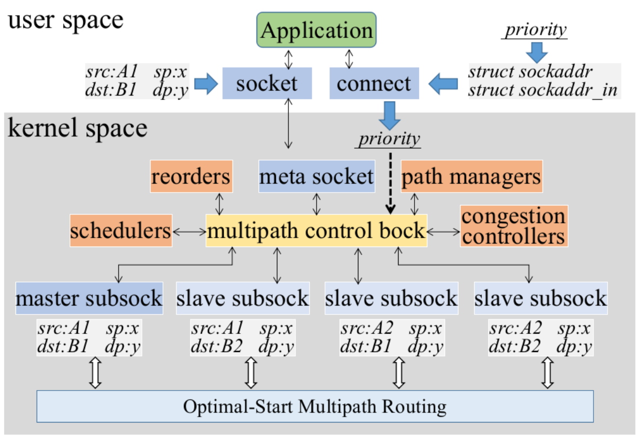


FIGURE 6. The relationship between each portion of MPTCP-L.

creates a TCP connection, several data structures are created in the kernel space: a meta socket, MPCB and subflows socket (master/slave subsocks). MPTCP-L includes three main components that are not standardized by the IETF: path managers, congestion controllers and schedulers.

Path managers manage the paths between the sender and the receiver. An outstanding characteristic of MPTCP is its ability to use several paths (subflows) in a single MPTCP connection. The path managers are responsible for creating and deleting subflows and reacting to events (activation or deactivation of network interfaces) by creating/removing the appropriate subflows. Current path managers that are built into the Linux MPTCP kernel implementation include default, fullmesh, ndiffports and binder. In this paper, we use the combination of improved MPTCP and improved MDR to help latency-sensitive applications to achieve high QoE. The combination is discussed in Section V.

Congestion Controllers: The congestion window (CWND) is used in the sender's flow control and is based on the network capacity and conditions. It is ordinarily in multiples of maximum segment size (MSS). The CWND is initially increased by TCP slow start. Once the cwnd reaches the slow start threshold or there is data loss due to congestion, and the cwnd growth changes to a congestion avoidance algorithm. Ultimately, the cwnd increases to either the network's limit due to congestion or it hits the receiver's window limit. Even though the rwnd (receive window) is larger than the cwnd (congestion window), the sender is bound by the cwnd. Therefore, the quantity of data the sender can send is min(rwnd, cwnd). The cwnd gives the limit of in-flight bytes, which is the quantity of data that has been sent but not yet acknowledged. The chosen congestion controller has an impact on the performance of MPTCP. The following three principles must be followed in the design of congestion control in MPTCP: (1) MPTCP's throughput should reach the throughput of the best subflow in all subflows; (2) MPTCP and traditional TCP should have the same throughput at the connection level, instead of each subflow having the same throughput as the traditional TCP at the subflow level;

and (3) MPTCP should choose the better subflow and move traffic away from the most congested paths. Currently available controllers built into the Linux MPTCP kernel implementation include LIA, OLIA, wVegas and BALIA. The default controller in the Linux MPTCP implementation is LIA.

A. SEND BUFFER AND RECEIVE BUFFER

MPTCP uses window-based data flow and congestion control similar to TCP but modified to accommodate multiple paths and the principles of MPTCP. The sender transfers segments over several subflows. The receiver aggregates subflows and reassembles the segments that come from different paths. However, it is challenging to handle buffer as send and receive buffers are shared among all subflows. On the receiver side, the received segments are stored in the receive buffer; only when all segments have been received in their correct sequence can they be passed to the appropriate application. In the sender side, segments that are being transmitted and are not acknowledged by the receiver are stored in the send buffer. That is, only the segments are acknowledged by a cumulative ACK, and they can be removed safely from the send buffer.

The quantity of data that can be sent is the minimum of the receive window (rwnd) and the congestion window (cwnd). The rwnd denotes how much data the receiver can advertise to the sender and can receive and buffer. The rwnd is also representative of the free buffer space for the socket. The amount of free space in the receive buffer is advertised to the sender in every ACK packet as the Window Size. The quantity of data the sender can send is more complicated. The upper bound is the receiver's advertised window, and the sender cannot send more than that, or data will be discarded. In addition, the sender cannot send more data at one time than is available in the send buffer, which is the buffer that the application writes data for TCP to send. The send buffer size is the size of the socket send buffer. The optimal send buffer size depends on the bandwidth-delay product (BDP), which denotes how much data the network can buffer. To achieve the highest throughput, it is vital to retain plenty of outstanding data (sent but not acknowledged) in the MPTCP connection. The ideal value (IV) for the outstanding data to achieve the best throughput for the MPTCP connection is a function of the BDP and the rwnd of each subflow. As shown in the following formula:

$$\mathrm{IV} = \min(\mathrm{BDP}, \text{total of all subflow rwnd})$$

If the IV value for the MPTCP connection is larger than the send buffer size, then the throughput achieved on the connection will not be optimal. Therefore, the size of the send buffer should be larger than the IV value.

Due to the difference in the characteristics of subpaths, segments transmitted to a fast path and sent later than segments transmitted to other paths can be out-of-order in the receiver's buffer. In addition, the so-called head-of-line blocking problem can occur when segments transmitted in fast subpaths arrive at the receiver and fill the receiver's buffer while waiting for segments transmitted in the slow subpaths. This issue is known as receive buffer blocking. Mechanisms such as opportunistic retransmission [47], nonrenegable selective acknowledgments [48], and smart scheduling decisions [14] are necessary to avoid these issues. However, in any case, the buffers must be large enough to deal with the maximum RTT of any subpaths. The buffer size requirements strongly depend on the RTTs of paths. In the traditional Internet, timer-based retransmissions should be rare; however, they must be considered due to the characteristics of MANETs. Hence, to utilize a network path and cover fast retransmission and timer-based retransmission, the send/receive buffer size constraint **B** is shown in the following formula [42]. In the worst case, it takes three times the highest subflow RTT (first transmission, fast retransmission, timer-based retransmission) plus the highest subflow RTO (retransmission timeout).

$$\mathrm{B} \geq \left(3 * \max_{1\leq i\leq n}(RTT_i) + \max_{1\leq i\leq n}(RTO_i)\right) * \sum_{i=1}^{N} Bandwidth_i$$

From the above formula, we can easily know when MPTCP simultaneously uses multiple paths with different delay characteristics; the required buffer size can be very large.

B. QoE-DRIVEN PACKET SCHEDULING

As shown in Fig. 6, MPTCP schedulers are responsible for transparently passing user data from the application layer over different active subflows and adding sequence numbers and confirmation numbers into segments before handing them to the network layer. When the receiver receives the segments, the schedulers reorder them and pass them to the application layer. To reorder packets, two levels of sequence numbers are used: the regular TCP sequence numbers that are used to ensure in-order delivery at the subflow level, and the data sequence numbers that can be used to guarantee in-order delivery at the MPTCP connection level. The scheduler has a large impact on the performance of data transmission. There are different ways to schedule the transmission of data. Current schedulers in the Linux kernel implementation include default, round-robin and redundant.

The different schedulers are detailed in [47]. However, due to the error-prone nature of wireless links and the high mobility of mobile nodes in MANETs, it is challenging to provide the required QoS for the end-users in MANETs. Therefore, the MPTCP schedulers in MANETs are wanted to manage real-time communication, such as audio and video. Nevertheless, the original MPTCP schedulers have some defects and cannot fit multiple kinds of applications to maximize throughput and reliability in MANETs. The fundamental problems with the default scheduler are threefold. (1) It always tries to transmit data over the subflow with the lowest RTT, as long as there is free space in the congestion window. Although it is less influenced by slow subflows than the round-robin scheduler, it can cause the aforementioned problems of head-of-line blocking and receive buffer blocking. The authors in [49] provided detailed classifications of the

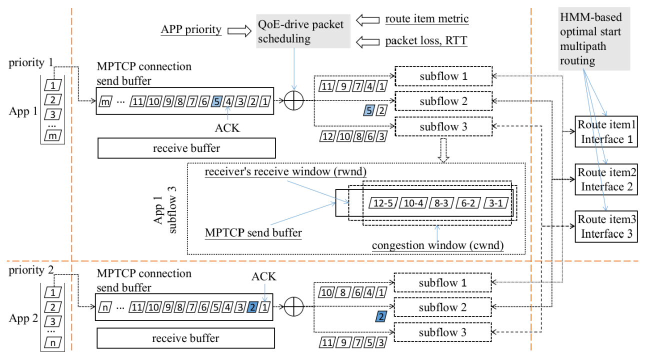


FIGURE 7. The framework of QoE-driven packet scheduling.

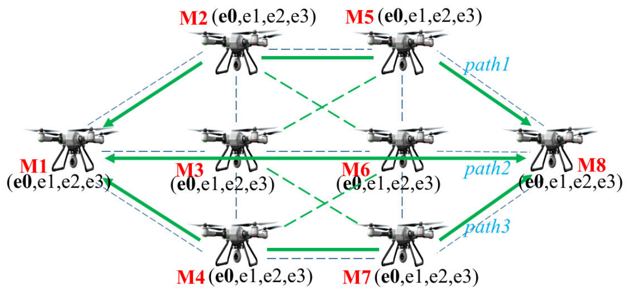


FIGURE 8. The application scenario of MPTCP in MANETs.

blocking issues. (2) When the packet is lost in multipath transmission, the scheduler must decide whether to retransmit this packet over the same subflow or over different ones [1]. (3) Lower priority applications may suffer from starvation.

To solve the above issues, the default scheduler includes a mechanism that reinjects packets, causing head-of-line blocking in a different subflow. The authors in [50] introduced a chunk rescheduling mechanism that reinjects the segment causing head-of-line blocking on a different subflow that has space available in its congestion window. The authors in [15] presented a delay aware packet scheduler that tries to send packet sequences in a manner that guarantees in-order delivery at the receiver. However, it is hard to guarantee QoS and higher QoE for real-time communications in MANETs. We thus propose the framework of QoE-driven packet scheduling, as shown in Fig. 7. We study the coordination between MPTCP and multipath routing in MANETs. We modify the MPTCP scheduler and path management based on optimal-start multipath routing. **Appendix B** shows how it works. We consider both the QoS of the subflows and the application priorities aiming to prevent applications with minimal QoS requirements from occupying more bandwidth.

To solve the issues of out-of-order, head-of-line blocking and receive buffer blocking, our scheme distributes packets among subflows according to the rwnd size, the cwnd size, bandwidth, RTT and priority. For instance, subflows with a smaller RTT are allocated more packets, more packets belonging to an application with higher priority can be scheduled to subflows as much as possible, and as long as there is free space in the rwnd and cwnd, packets can be scheduled to subflows. Due to the characteristics of MANETs, the BDP may not be a constant. Hence, packets should be distributed among the subflows proportional to the BDP of the subflows aiming to adapt well to fluctuating network conditions caused by changes in BDP or subflow failures. Our QoE-driven packet scheduling algorithm is described in **Appendix B**. The source code for the scheduling algorithm is available from [51].

The related references mentioned earlier in this paper improved MPTCP's throughput and stability mainly by improving path manager and congestion controller. However, they did not consider both the method of adding routing table entries and the method of establishing MPTCP subpaths simultaneously. As far as we know, this is the first time that the method of adding routing table entries and the method of establishing MPTCP subpaths have been simultaneously considered to offer more efficient use of multiple subpaths and better network traffic load balancing to increase throughput and reliability. MPTCP doesn't have the ability to update the routing table, and is only based on routing table for path manager. The improved OSPF-MDR is responsible for updating the routing table. MPTCP and OSPF-MDR are loosely coupled, as shown in Fig. 7.

V. IMPLEMENTATION, TESTING AND EVALUATION

A. IMPLEMENTATION OF MPTCP IN MANETS

To evaluate the performance of MPTCP in MANETs, a simplified topology is more useful than a complex topology. Fig. 8 shows a scenario of MPTCP in MANETs. We assume that each node (M1–M8) is equipped with four network interfaces (e0,e1,e2,e3). IP addresses assigned to network interfaces belong to different network segments. In expression 1, Y denotes nodes (M1–M8), X denotes interfaces (e0,e1,e2,e3).

$$112.26.X.Y/24, \quad X \in [0,3], \; Y \in [1,8] \tag{1}$$

1) HMM-BASED OPTIMAL-START MULTIPATH ROUTING

In this paper, MPTCP mainly relies on HMM-based optimal-start multipath routing to hash different subflows to different paths. In the scenario shown in Fig. 8, MDR [52] can only generate routing information for e0; however, MPTCP needs to build subpaths using interfaces (e0,e1,e2,e3); therefore, MDR cannot provide routing support for the establishment of the MPTCP subpaths. To solve this problem, it is necessary to improve MDR; the method is as follows: generate routing table entries for e0; meanwhile, the same routing table entries are also generated for e1,e2,e3. For example, in M1, the routing table entries to M8 generated by the original MDR are represented in equation 2, where E_0RE represents the routing table entries for e0 in M1.

$$E_0RE = nexthop\ via\ 112.26.0.Y\ eth0\ to\ 112.26.0.8, \quad Y \in [2,4] \tag{2}$$

$$E_{XY}RE = nexthop\ via\ 112.26.0.Y\ eth0\ to\ 112.26.X.8,\quad X \in [1,3], Y \in [2,4] \tag{3}$$

$$E_{X}RE = nexthop\ via\ 112.26.0.Y\ eth0\ to\ 112.26.X.8,\quad X \in [1,3], Y = X+1 \tag{4}$$

After the first improvement to MDR, in M1, the routing table entries to M8 generated by the improved MDR, in addition to the three entries (as represented in equation 2), include the newly added 9 entries; that is, three routing table entries are generated for each destination address (e1,e2,e3), as represented in equation 3, where E_{XY}RE represents the routing table entries for e1,e2,e3 in M1. Now, if M1 can successfully ping e0 in M8, then M1 can successfully ping e1,e2,e3 in M8.

To allow the routing table to provide better network load balancing for multipath transmission, a second improvement to MDR is necessary. The method is as follows: only one routing table entry is generated for each destination address (e1,e2,e3). Hence, in M1, there are 6 routing table entries to M8 generated by the second improved MDR, as shown in equations 2 and 4, where E_XRE represents the routing table entries for e1,e2,e3 in M1.

The improved multihop routing algorithm is described in **Appendix C**.

2) MPTCP ON ANDROID

It is the key to the implementation of MPTCP in MANETs to run MPTCP on MSDs. In this paper, we assume that the operating system on UAVs or MSDs is Android. We port MPTCP to Android-x86-nougat (Android 7.1, kernel 4.9.31) based on the MPTCP Linux kernel implementation. The documents for porting MPTCP to Android, all-modified source code files and MPTCP demo (mp4 file) can be obtained from [51].

3) IMPROVING SUBPATHS ESTABLISHMENT ALGORITHM

We assume that M1 acts as a server, M8 acts as a client, the original MPTCP uses M8:(e0,e1,e2,e3) and M1:(e0,e1,e2,e3) to establish subpaths, such as the MPTCP subpaths of M8 to M1, as represented in expression 5.

$$src:\ 112.26.X.8,\ dst:\ 112.26.Y.1,\quad X, Y \in [0,3] \tag{5}$$

$$src:\ 112.26.X.8,\ dst:\ 112.26.X.1,\quad X \in [0,3] \tag{6}$$

Why not use the subpaths as in expression 6? The root reason resides in the improvement to the routing protocol in Section V.A.1. For better network traffic load balancing, it is necessary to establish subpaths represented in expression 6. Hence, it is essential to improve the subpath establishment algorithm.

Table 3 shows the subpaths established by the improved MPTCP. NoNH is the number of next-hops. NoNH=3 of the master-subpath improves its connectivity reliability. The interfaces (M1-M8:e0) play three roles: (1) used by quagga to generate multihop routing entries; (2) used to establish the master-subpath; (3) midnodes to forward packets. The interfaces M1-M8:(e1,e2,e3) are used to establish slave subpaths.

TABLE 3. The established subpaths.

subpath	src-IP	dst-IP	middle nodes	NoNH
master-subpath	M8:e0	M1:e0	M2–M7	≤ 3
slave-subpath1	M8:e1	M1:e1	M2, M5	≤ 1
slave-subpath2	M8:e2	M1:e2	M3, M6	≤ 1
slave-subpath3	M8:e3	M1:e3	M4, M7	≤ 1

The improved algorithm for creating subflow is shown below. The key is to create subpaths using pairs of IP addresses that belong to the same network segment.

Algorithm for Creating Subflow

```
VAR: local_addr[i] ← local ip addresses
VAR: remote_addr[j]
VAR: struct mptcp_loc4 loc
VAR: struct mptcp_rem4 rem
 1: BEGIN:
 2:    PROCEDURE: create_subflow_worker
 3:       remote_addr[j] ← peer ip addresses & ports
 4:       for ip_src in local_addr[i]
 5:          for ip_dst in remote_addr[j]
 6:             if ip_src can syn ip_dst successfully &&
 7:                ( ip_src/24 == ip_dst/24 || ip_src/24 ==
"112.26.0.0")
 8:                rem.addr = ip_dst->addr
 9:                rem.port = ip_dst->port
10:                loc.addr = ip_src->addr
11:                loc.port = ip_src->random_port
12:                mptcp_v4_subflows(loc, rem)
13:             end if
14:          end for
15:       end for
16:    END PROCEDURE
17: END
```

B. TESTING AND EVALUATION

FEP [53] is used as the test platform.

1) TESTING GOALS

We use real-time data traffic to assess whether multipath communication is suitable for latency-sensitive applications and can help special applications to satisfy the requirements of end-users in MANETs. Multipath transport in heterogeneous networks is challenging. Therefore, we consider MPTCP and evaluate their proficiency in transporting real-time data traffic over homogeneous/heterogeneous networks.

We address three key testing goals: (1) achieve performance advantages over TCP; (2) offer robust performance; and (3) although there may be fear of an overly large buffer space need due to packet reordering over very dissimilar

TABLE 4. Transmission rates of interfaces in Homo/Hetero scenarios.

interface	homogeneous scenario	heterogeneous scenario
eth0	11 Mbps	11 Mbps
eth1	11 Mbps	5.5 Mbps
eth2	11 Mbps	2 Mbps
eth3	11 Mbps	1 Mbps

paths, we show that buffer size requirements remain reasonably small. This is particularly important for systems having to manage many simultaneous connections.

a: HOMOGENEOUS/HETEROGENEOUS SCENARIO

The test environment is an IBM server with 32 CPU-cores, 64 GB of memory, and Fedora 26 installed. We create eight VirtualBox instances for Android, each of which has four network interfaces. The network topology is generated by the NS-3 script [51], which is shown in Fig. 8 and has eight nodes representing eight mobile systems with four interfaces per mobile system. Table 4 shows the transmission rates of interfaces in homogeneous scenarios or heterogeneous scenarios. The scenarios are distinguished by setting different parameters for the network interfaces. In this way, a unified model can be used for testing; only the parameters are different.

b: MEASUREMENT TOOLS AND TEN COMBINATIONS OF PROTOCOLS

We use *iperf* and *nc* as measurement tools to generate real-time data traffic and record the throughput and RTTs of the paths during the bandwidth and delay measurements. Ten combinations of protocols are tested and compared: (1) TCP-MDR **(TM):** the combination of the standard TCP and the original MDR with multihop routing; (2) MPTCP-FullPath-MDR **(MFM):** the combination of the standard MPTCP [43] and the fullpath MDR using several (four) virtual network interfaces. The term FullPath indicates that full connection (one-to-many connection) among multiple network interfaces in adjacent nodes is established via a routing table generated by MDR; (3) MPTCP-FullPath-xIF-MDR **(MFxM):** the combination of the standard MPTCP and the fullpath (xIF) MDR. The term xIF indicates that several (four) real network interfaces are used; (4) MPTCP-PartPath-xIF-MDR **(MPxM):** the combination of the standard MPTCP and the partpath (xIF) MDR. The term partpath indicates that part connection (one-to-one connection) among multiple network interfaces in adjacent nodes is established via a routing table generated by MDR; (5) MPTCP-Markov-F-xIF-MDR **(MMFxM):** the combination of the standard MPTCP and the Markov fullpath (xIF) MDR. The term F is fullpath. The term Markov indicates that the HMM-based optimal-start multipath routing algorithm is used in MDR; (6) MPTCP-Markov-P-xIF-MDR **(MMPxM):** the combination of the standard MPTCP and the Markov partpath (xIF) MDR. The term P is partpath; (7) Partflow-MPTCP-Markov-F-xIF-MDR **(PMMFxM):** the combination of the improved MPTCP and the Markov fullpath (xIF) MDR. The term Partflow indicates that the subflow is established according to the transmission rates of interfaces; (8) Partflow-MPTCP-Markov-P-xIF-MDR **(PMMPxM):** the combination of the improved MPTCP and the Markov partpath (xIF) MDR.

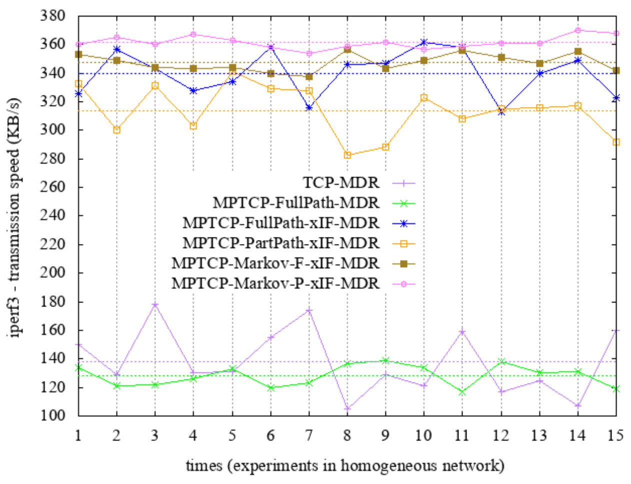


FIGURE 9. iperf3 - transmission speed and average transmission speed difference over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

2) EXPERIMENTS OF DIFFERENT PROTOCOLS IN HOMOGENEOUS SCENARIOS

We run command *iperf3* in M1 to make it a server, run command *iperf3* in M8 to download data from M1. *iperf3* runs for the 180 s to allow the flows to reach equilibrium. We carried out experiments 90 times to compare the data transfer performances of TM, MFM, MFxM, MPxM, MMFxM and MMPxM. The test results are shown in Figs. 8 and 9. Fig. 10 represents the results of box-and-whisker plots. To attain reliable results, each test runs for 180 s.

The test results show that MPTCP successfully runs in MANETs. Note that the topology is simplified by letting NS-3 nodes remain passive without affecting the function test.

As shown in Fig. 9, the dotted lines indicate the average transmission speeds. The average transmission speeds are different over TM (138 KBytes/s), MFM (128 KBytes/s), MFxM (340 KBytes/s), MPxM (313.8 KBytes/s), MMFxM (347.4 KBytes/s) and MMPxM (361.6 KBytes/s). (1) In the single-interface case, for the transmission speed, MFM cannot provide a better throughput than TM. The reason is the reordering caused by retransmissions on the lossy paths. With the default buffer size, the full performance is already reached. This is similar to the size needed for TCP. In addition, MPTCP increases the delay compared to TCP as the data are split among subflows. (2) In the multi-interface case, due to using four paths, the transmission speed over MFxM, MPxM, MMFxM and MMPxM is two to three times the transmission speed over TM and MFM. MMPxM has the highest transmission speed.

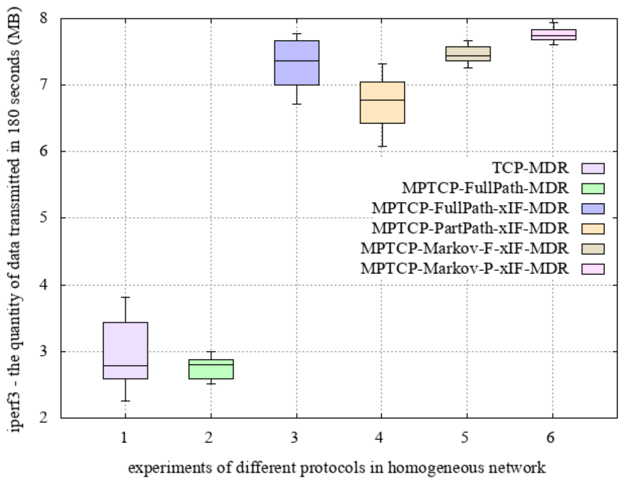


FIGURE 10. The quantity of data transmitted in 180 s (MB) over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

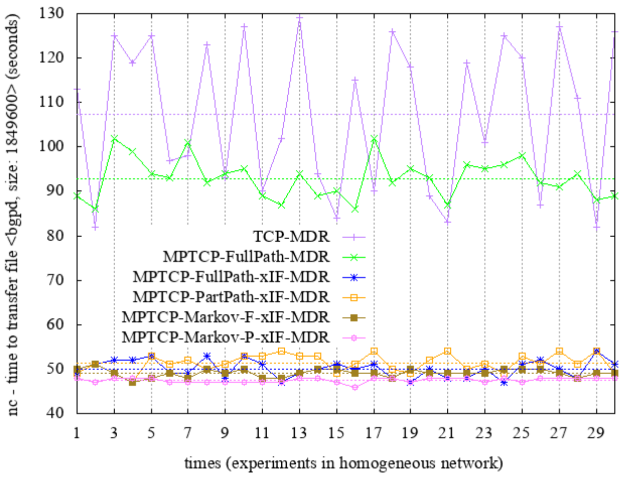


FIGURE 11. nc - time to transfer file and average transmission time difference over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

As shown in Fig. 10, MMFxM and MMPxM have smaller fluctuations than others due to more stable data transmission. Therefore, MMFxM and MMPxM achieve higher reliability. TM has the largest fluctuation for the quantity of data transmitted in 180 s. In addition, MMFxM and MMPxM also have better aggregation benefits. MMPxM has the largest quantity of data transmitted in 180 s, which is more than MFxM, MPxM and MMFxM. In addition, MMPxM has the smallest fluctuation due to reducing the impact of network connection changing problem by using partial paths and Markov. The fluctuation of MMFxM is similar to MMPxM. The fluctuation of MFxM is similar to that of MPxM; however, the quantity of data transmitted by MFxM is greater than that of MPxM. In addition, the fluctuation difference between MMFxM and MMPxM is small. Specifically, MMPxM has the largest steady throughput and the smallest fluctuation.

We run *nc* in M1 as a file server, and run command *nc* in M8 to download a file *(quagga/sbin/bgpd, just a binary file with a fixed size of 1.8 MB)* from M1. We carried out experiments 180 times to compare the data transfer performances of TM, MFM, MFxM, MPxM, MMFxM and MMPxM. The test results are shown in Figs. 10–13.

As shown in Fig. 11, the dotted lines indicate the average transmission time. The average transmission times are different over TM (107.33 s), MFM (92.93 s), MFxM (50.07 s), MPxM (51.33 s), MMFxM (49.1 s) and MMPxM (47.53 s). (1) in the single-interface case, for the transmission time, TM is worse than MFM. The reason is that MFM can send the file via multiple subflows over one path. (2) In the multi-interface case, due to using four paths, the transmission time over MFxM, MPxM, MMFxM and MMPxM is a third to a quarter of the transmission time over TM and MFM. MMPxM has the least transmission time.

As shown in Fig. 12, MMFxM and MMPxM have smaller fluctuations than the other methods. Therefore, MMFxM and MMPxM achieve higher reliability. TM has the largest

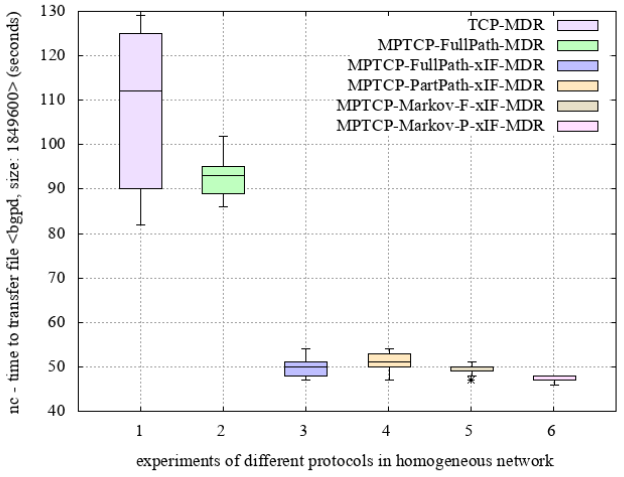


FIGURE 12. nc - time to transfer file and average transmission time difference over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

fluctuation for the transmission time. MFxM, MPxM, MMFxM and MMPxM have similar average transmission times, which means they can transfer the file over multiple paths and obtain bandwidth aggregation. However, MMPxM has the least average transmission time. In addition, MMFxM and MMPxM have the smallest fluctuation due to reducing the impact of network connection changing problem using Markov. The fluctuation of MMFxM is similar to MMPxM. The fluctuation of MFxM is similar to MPxM. Thus, the test results show that multipath transmission can effectively increase the application payload throughput and greatly improve the robustness of the data transmission, mainly due to the use of multiple paths and Markov.

As shown in Fig. 13, the red dotted line indicates that the exact number (2,586) of packets should be transferred from M1 to M8. M1 sent approximately 5,600 to 6,000 packets, and 3,000 to 3,400 of the packets were duplicated. Let us look at Fig. 8 again, and the path from M1 to M8 is [M1] → [M2, M3 or M4] → [M5, M6 or M7] → [M8]. For every

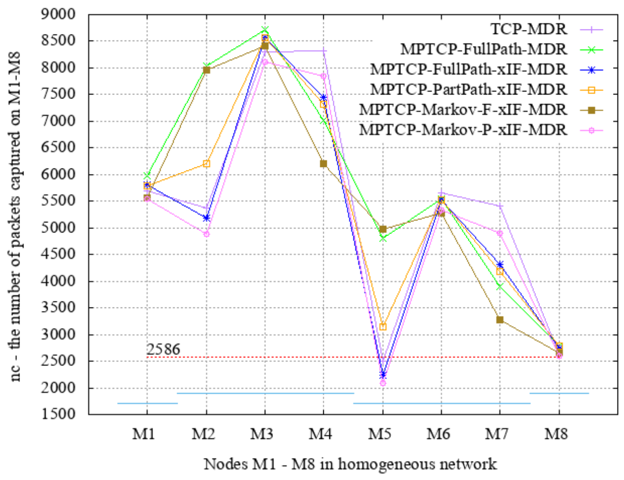


FIGURE 13. The number of packets captured on M1–M8 over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

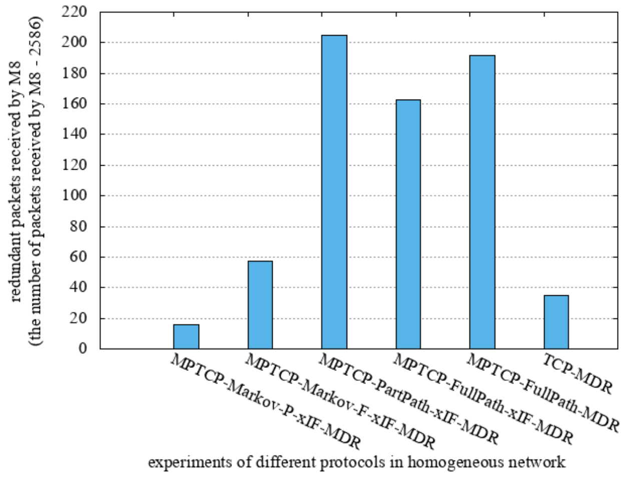


FIGURE 14. The number of packets received by M8 - 2586 over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

test of TM, MFM, MFxM, MPxM, MMFxM and MMPxM, M3 captured more packets than M2 and M4, M6 captured more packets than M5 and M7 because the path from M1 to M3 and the path from M6 to M8 were shorter than others. Finally, M8 received approximately 2,600 to 2,800 packets.

When M8 is downloading the file from M1, the command *tcpdump* is used to capture packets received on each network interface (eth0, eth1, eth2 and eth3) of M1–M8. Fig. 14 shows the difference in the number of redundant packets received by M8 when testing TM (35), MFM (192), MFxM (163), MPxM (205), MMFxM (57) and MMPxM (16). It is obvious that MMPxM received the least redundant packets than the others. MMFxM and MMPxM received less redundant packets than MFM, MFxM and MPxM due to the use of Markov.

3) EXPERIMENTS OF DIFFERENT PROTOCOLS IN HETEROGENEOUS SCENARIOS

We run command *iperf3* in M1 to make it a server, run command *iperf3* in M8 to download data from M1. *iperf3* runs for 180 s to allow the flows to reach equilibrium.

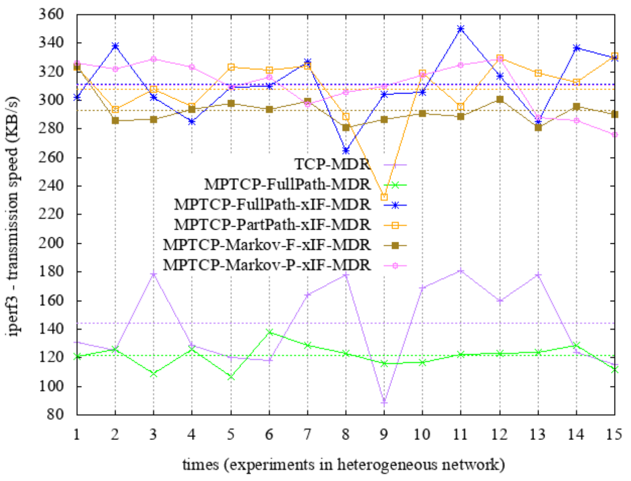


FIGURE 15. iperf3 - transmission speed and average transmission speed difference over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

We carried out experiments 90 times to compare the data transfer performances of TM, MFM, MFxM, MPxM, MMFxM and MMPxM. The test results are shown in Figs. 14 and 15. Fig. 16 represents the results of box-and-whisker plots. To attain reliable results, each test runs for 180 s.

The test results show that MPTCP is successfully running in MANETs. As shown in Fig. 15, the dotted lines indicate the average transmission speeds. The average transmission speeds are different over TM (143.97 KBytes/s), MFM (121.47 KBytes/s), MFxM (311.13 KBytes/s), MPxM (307.87 KBytes/s), MMFxM (293.2 KBytes/s) and MMPxM (310.67 KBytes/s). (1) In the single-interface case, for the transmission speed, MFM cannot provide better throughput than TM because of the reordering caused by retransmissions on the lossy paths. With the default buffer size, the full performance is already reached. This is similar to the size needed for TCP. In addition, MPTCP increases the delay compared to TCP as the data are split among subflows. (2) In the multi-interface case, due to using four paths, the transmission speed over MFxM, MPxM, MMFxM and MMPxM is two to three times the transmission speed over TM and MFM. MFxM has the highest transmission speed. However, MMPxM has a similar average transmission speed as MFxM.

As shown in Fig. 16, TM has the largest fluctuation for the quantity of data transmitted in 180 s. MMFxM has the least fluctuation, and the fluctuation of MMFxM is similar to MFM. Therefore, MMFxM achieves higher reliability. In addition, MFxM, MPxM, MMFxM and MMPxM have better aggregation benefits by using multiple paths. However, the fluctuation difference between MMFxM and MMPxM is larger. The performance of MMFxM in heterogeneous scenarios is more stable than that of MMPxM.

We run command *nc* in M1 to make it a file server, run command *nc* in M8 to download the file from M1. We carried out experiments 180 times to compare the data transfer performances of TM, MFM, MFxM, MPxM, MMFxM and MMPxM. As shown in Fig. 17, the red dotted line indicates

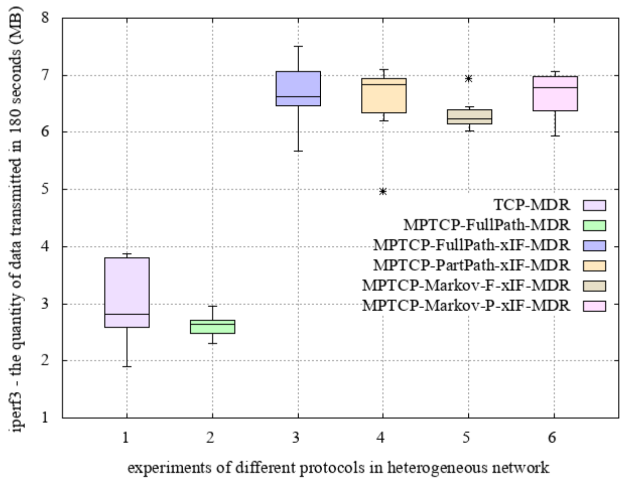


FIGURE 16. The quantity of data transmitted in 180 s (MB) over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

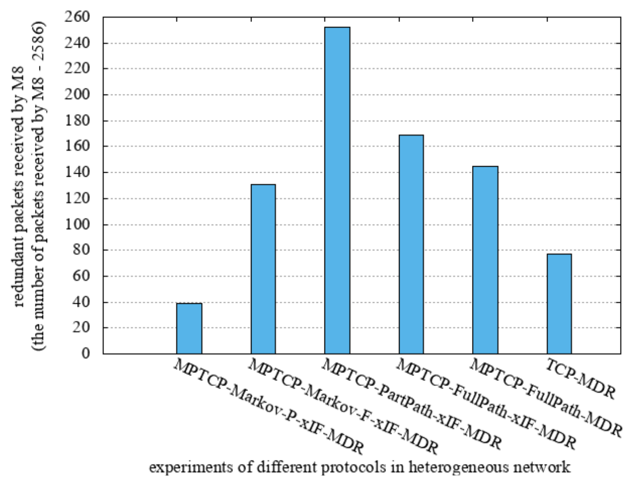


FIGURE 18. The number of packets received by M8 - 2586 over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

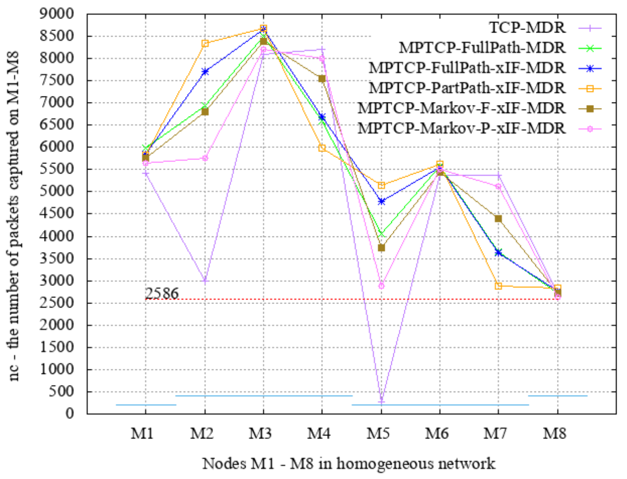


FIGURE 17. The number of packets captured on M1–M8 over TM, MFM, MFxM, MPxM, MMFxM and MMPxM.

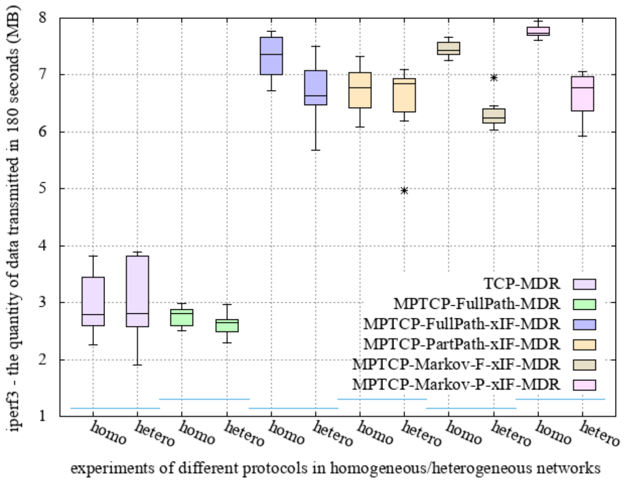


FIGURE 19. The quantity of data transmitted in 180 s (MB) over TM, MFM, MFxM, MPxM, MMFxM and MMPxM in various network scenarios (homogeneous and heterogeneous).

that the exact number (2,586) of packets should be transferred from M1 to M8. M1 sent approximately 5,400 to 6,000 packets, and 2,800 to 3,400 of the packets were duplicated. Let us look at Fig. 8 again, and the path from M1 to M8 is [M1] → [M2, M3 or M4] → [M5, M6 or M7] → [M8]. For every test of TM, MFM, MFxM, MPxM, MMFxM and MMPxM, M3 captured more packets than M2 and M4, M6 captured more packets than M5 and M7 because the path from M1 to M3 and the path from M6 to M8 are shorter than the others. Finally, M8 received approximately 2,600 to 2,800 packets.

When M8 is downloading the file from M1, the command *tcpdump* is used to capture packets received on each network interface (eth0, eth1, eth2 and eth3) of M1–M8. Fig. 18 shows the difference in the number of redundant packets received by M8 over TM (77), MFM (145), MFxM (169), MPxM (252), MMFxM (131) and MMPxM (39). It is obvious that MMPxM received the least redundant packets than others. In addition, MMPxM can achieve better results than MMFxM when using Markov.

As shown in Fig. 19, there is not a large difference in the quantity of data transmitted in homogeneous and heterogeneous scenarios for TM, MFM, MFxM and MPxM. However, for MMFxM and MMPxM, the quantity of data transmitted in the homogeneous scenario is more than that in the heterogeneous scenario because the transmission rates of interfaces in homogeneous areas are the same, namely, 11 Mbps. However, the transmission rates of interfaces in heterogeneous areas are 11, 5.5, 2 and 1 Mbps.

4) DIFFERENT BUFFER SIZES OF DIFFERENT PROTOCOLS IN HETEROGENEOUS SCENARIOS

To determine the receive buffer size (*net.ipv4.tcp_rmem*), we perform ping tests from M8 to M1 (*ping -c 100 112.26.0.1*) three times. The RTT values are as below.

min/avg/max/mdev=11.733/81.778/552.873/68.240 ms
min/avg/max/mdev=21.392/84.480/347.809/55.355 ms
min/avg/max/mdev=12.332/75.246/365.917/51.515 ms

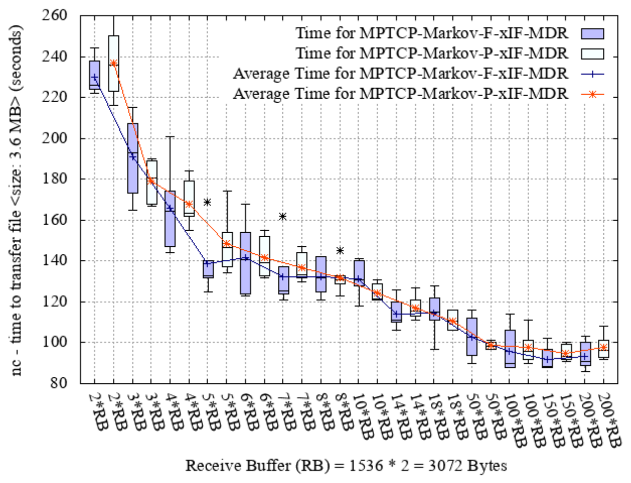


FIGURE 20. Time to transfer file <libstlport_shared.so, 3.6 MB> in various network scenarios (different buffer size) versus various values of net.ipv4.tcp_rmem.

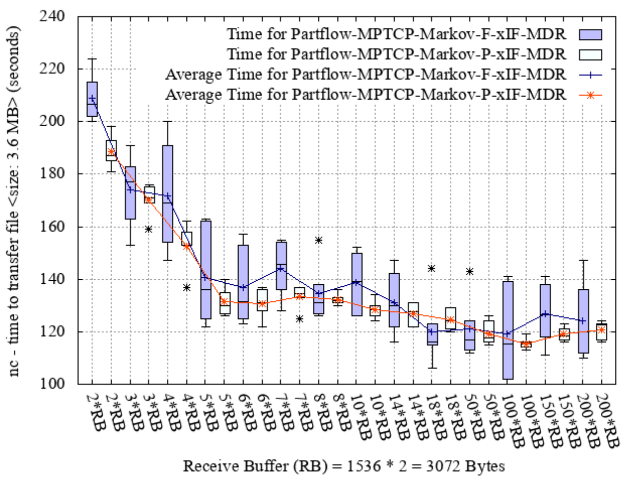


FIGURE 21. Time to transfer file <libstlport_shared.so, 3.6 MB> in various network scenarios versus various values of net.ipv4.tcp_rmem.

average: (transfer 5.534 MBytes, bandwidth 257.867 Kbits/s)

Thus, BDP = bandwidth * RTT / 2 = 257.867 * 0.084480 / 2 = 10.89230208 Kbits ~ = 10892 bits = 1361.5 Bytes,

receive_win = max [1361.5, 1024+512] = 1536 Bytes,

Due to *net.ipv4.tcp_adv_win_scale = 1*,

Therefore, *receive_win = net.ipv4.tcp_rmem / 2*,

Set ***RB*** *= 2 * receive_win = 3072 Bytes*.

Note: mdev denotes mean deviation. BDP denotes the bandwidth-delay product. **RB** represents the receive buffer.

We run command *nc* in M1 to make it a file server, run command *nc* in M8 to download the file *libstlport_shared.so (3.6 MB)* from M1. We carried out experiments 180 times to compare the data transfer performances of MMFxM, MMPxM, PMMFxM and PMMPxM. The test results are shown in Figs. 19 and 20.

We investigated the effect of buffer size on the MPTCP. The buffer size varies between 2*RB and 200*RB. MPTCP uses multiple paths with different delay characteristics.

As shown in Fig. 20, the average transmission time rapidly decreases as *net.ipv4.tcp_rmem* increases when *net. ipv4.tcp_rmem* is less than 6*RB for MMFxM and MMPxM. As shown in Fig. 21, the average transmission time rapidly decreases as *net.ipv4.tcp_rmem* increases when *net.ipv4.tcp_rmem* is less than 6*RB for PMMFxM and PMMPxM. PMMPxM has smaller fluctuations than PMMFxM.

As shown in Fig. 22, the average transmission time rapidly decreases as *net.ipv4.tcp_rmem* increases when *net.ipv4.tcp_rmem* is less than 6*RB for MMFxM and PMMFxM. When *net.ipv4.tcp_rmem* is more than 6*RB, MMFxM has less average transmission time than PMMFxM.

As shown in Fig. 23, the average transmission time rapidly decreases as *net.ipv4.tcp_rmem* increases when *net.ipv4.tcp_rmem* is less than 6*RB for MMPxM and PMMPxM. When *net.ipv4.tcp_rmem* is more than 6*RB, MMPxM has less average transmission time than PMMPxM.

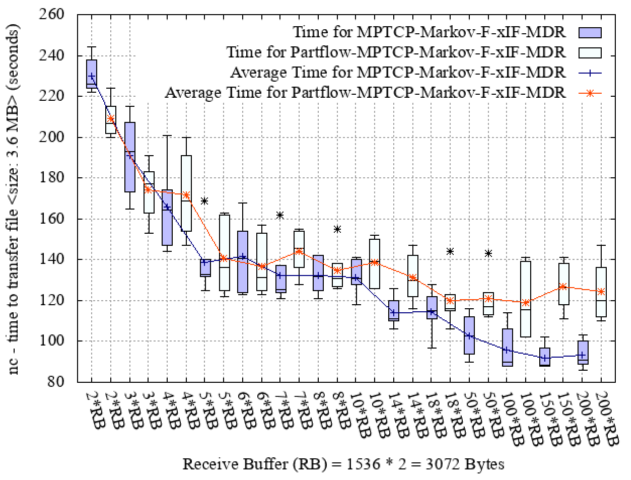


FIGURE 22. Time to transfer file <libstlport_shared.so, 3.6 MB> in various network scenarios versus various values of net.ipv4.tcp_rmem.

In addition, when *net.ipv4.tcp_rmem* is less than 6*RB, PMMPxM has less average transmission time and smaller fluctuation than MMFxM.

Four factors influence the choice of path: (1) the priority of this path to the end machine, (2) the priority of this path in the local machine, (3) whether the packet has been sent using this path, the packet cannot be sent repeatedly on the same path, and (4) SRTT (smoothed RTT), which is calculated by equation 7, and RTO, which is calculated by equation 8.

The adjustment of path priority can be made through the commands: *ip link set dev eth0 multipath backup*.

$$SRTT = (ALPHA * SRTT) + ((1 - ALPHA) * RTT) \quad (7)$$

$$RTO = min[TCP\_RTO\_MAX, max[TCP\_RTO\_MIN, (BETA * SRTT)]] \quad (8)$$

ALPHA is a smoothing factor, e.g., 1/8. BETA is a delay variance factor, e.g., 2.

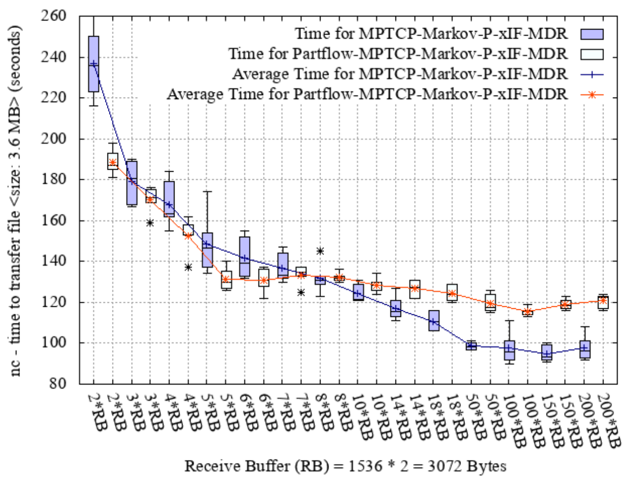


FIGURE 23. Time to transfer file <libstlport_shared.so, 3.6 MB> in various network scenarios. Contrast between different protocols.

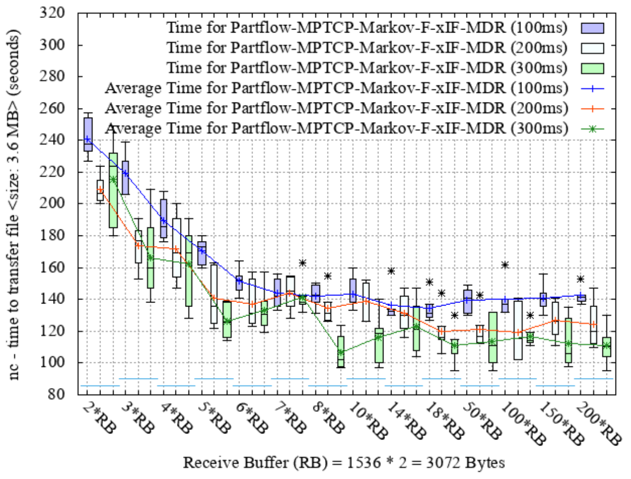


FIGURE 24. Time to transfer file <libstlport_shared.so, 3.6 MB> in various network scenarios (different buffer size, different RTO) versus various values of net.ipv4.tcp_rmem.

We mainly adjust the TCP_RTO_MIN to affect the scheduling of packets. The TCP_RTO_MIN is set to 100 ms, 200 ms, and 300 ms for PMMFxM100, PMMFxM200 and PMMFxM300, respectively, in the file *kernel/include/net/tcp.h* as follows:

#define TCP_RTO_MIN ((unsigned)(HZ/10))

#define TCP_RTO_MIN ((unsigned)(HZ/5))

*#define TCP_RTO_MIN ((unsigned)(3*HZ/10))*

We run command *nc* in M1 to make it a file server, run command *nc* in M8 to download the file *libstlport_shared.so (3.6 MB)* from M1. We carried out experiments 3*14*6=252 times to compare the data transfer performances of PMMFxM100, PMMFxM200 and PMMFxM300. The test results are shown in Fig. 24.

In Fig. 24, the average transmission time is rapidly decreased as *net.ipv4.tcp_rmem* increases when *net.ipv4.tcp_rmem* is less than 6*RB for PMMFxM100, PMMFxM200 and PMMFxM300. As shown in Fig. 24, the average

TABLE 5. Percentage of packets received of each protocol in Homogeneous scenarios.

	TM	MFM	MFxM	MPxM	MMFxM	MMPxM
M1.eth0	5619	5969	1510	1709	1441	1577
M8.eth0	2585	2778	732	842	681	756
eth0.rec	0.46	0.4654	0.4848	0.4927	0.4726	0.4794
M1.eth1	0	0	1476	1182	1388	1253
M8.eth1	0	0	701	558	661	574
eth1.rec	0	0	0.4749	0.4721	0.4762	0.4581
M1.eth2	0	0	1246	1184	1388	1190
M8.eth2	0	0	563	585	654	543
eth2.rec	0	0	0.4518	0.4941	0.4712	0.4563
M1.eth3	0	0	1578	1710	1352	1516
M8.eth3	0	0	753	806	647	729
eth3.rec	0	0	0.4772	0.4713	0.4786	0.4809

transmission time of PMMFxM300 is less than that of PMMFxM200, and PMMFxM200 is less than that of PMMFxM100. The TCP buffer size (receive buffer) is set to x*RB (x values are 2, 3, 4, 5, 6, 7, 8, 10, 14, 18, 50, 100, 150, 200). Side-by-side comparisons are made between PMMFxM100, PMMFxM200 and PMMFxM300 to study the influence of the receive buffer size (x*RB) on the stability and performance improvement. Fig. 24 shows how the different values of the receive buffer size in a heterogeneous scenario influence the time of the file sending. Therefore, it is important to adjust *TCP_RTO_MIN* according to the RTT.

As seen in Figs. 19–23, when *net.ipv4.tcp_rmem* is less than 6*RB, the average transmission time rapidly decreases as *net.ipv4.tcp_rmem* increases in all cases, which means that the receive buffer size is severely insufficient for transferring packets from a delay perspective. When *net.ipv4.tcp_rmem* is larger than 6*RB, the average transmission time slowly decreases as *net.ipv4.tcp_rmem* increases.

5) PERCENTAGE OF PACKETS RECEIVED AND END-TO-END DELAY OF EACH PROTOCOL IN HOMO/HETEROGENEOUS SCENARIOS

Not all protocols can achieve full throughput due to lossy paths. Table 5 and Table 6 show how much each link received. M1.eth0 indicates the number of packets sent by node M1 through eth0, M8.eth0 indicates the number of packets received by node M8 through eth0, eth0.rec indicates the percentage of packets on the path between node M1 and node M8; therefore, the packet loss rate is *1 - eth0.rec*, and so on. As shown in Table 5 and Table 6, the packet loss rates range from 50% to 56%.

Table 5 provides statistics on the six protocols: TM, MFM, MFxM, MPxM, MMFxM and MMPxM. As seen in Table 4, all interfaces in homo scenarios have the same transmission rate, and the number of sending and receiving packets in the four paths is similar.

TABLE 6. Percentage of packets received of each protocol in Heterogeneous scenarios.

	TM	MFM	MFxM	MPxM	MMFxM	MMPxM	PMMFxM	PMMPxM
M1.eth0	5338	5994	2134	2277	2163	2145	3062	3528
M8.eth0	2663	2731	1037	1122	1002	1026	1414	1713
eth0.rec	0.4989	0.4556	0.4859	0.4928	0.4632	0.4783	0.4618	0.4855
M1.eth1	0	0	1907	1778	1790	1667	2486	1992
M8.eth1	0	0	890	880	841	799	1175	946
eth1.rec	0	0	0.4667	0.4949	0.4698	0.4793	0.4726	0.4749
M1.eth2	0	0	1262	1049	1123	1202	0	0
M8.eth2	0	0	588	497	520	564	0	0
eth2.rec	0	0	0.4659	0.4738	0.463	0.4692	0	0
M1.eth3	0	0	539	731	578	738	0	0
M8.eth3	0	0	240	339	262	328	0	0
eth3.rec	0	0	0.4453	0.4637	0.4533	0.4444	0	0

Table 6 provides statistics on the eight protocols: TM, MFM, MFxM, MPxM, MMFxM, MMPxM, PMMFxM and PMMPxM. As seen from Table 4, the interfaces (eth0, eth1, eth2, eth3) in the heterogeneous scenarios have different transmission rates, and the number of sending and receiving packets in the path between M1.eth0 and M8.eth0 is significantly higher than in other paths. Due to the different transmission rates of interfaces, two tests (PMMFxM and PMMPxM) are performed in heterogeneous scenarios but not in homogeneous scenarios. In the heterogeneous scenarios, two subflows are established according to transmission rates of interfaces.

Because the main application scenario of the model proposed in this paper is real-time communication, for example, VoIP, we give an evaluation for the end-to-end delay of each protocol in different scenarios. Figs. 24 and 25 show the time from sending a packet to receiving its acknowledgment packet and average delay on each pair of interfaces (eth0, eth1, eth2, eth3) between node M1 and node M8 for each protocol in the homogeneous and heterogeneous scenarios, respectively.

As shown in Fig. 25, MMPxM sends the same quantity of data in less time than other protocols. The delay and fluctuation of subflows created on different interfaces are different, subflow created on eth0 has the least delay and the best stability, subflow created on eth3 has the same delay and stability as subflow created on eth0 after 36 s, as shown in Table 5. The number of valid packets successfully transmitted by the two subflows is similar (756 and 729). MMFxM sends the same quantity of data in a time similar to MMPxM, and the delay and fluctuation of subflows created on different interfaces are similar. As shown in Table 5, the number of valid packets successfully transmitted by the four subflows is similar.

As shown in Fig. 26, MMFxM sends the same quantity of data in less time than other protocols, and the delay and fluctuation of subflows created on different interfaces are different. As shown in Table 6, the number of valid packets successfully transmitted by the subflows is

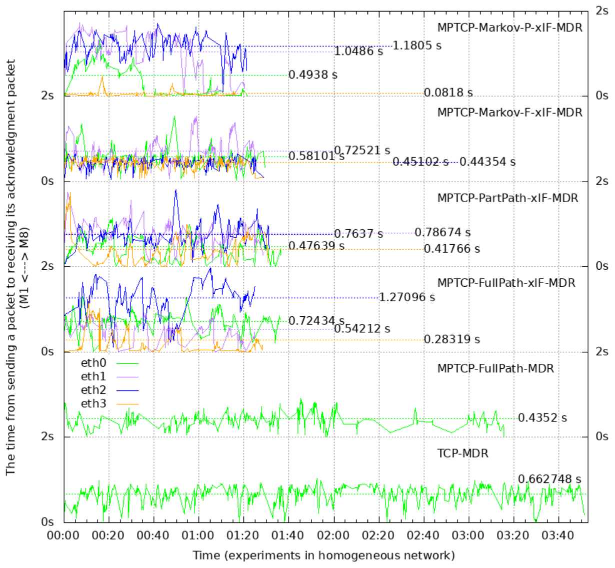


FIGURE 25. The end-to-end delay of each protocol in homogeneous scenarios.

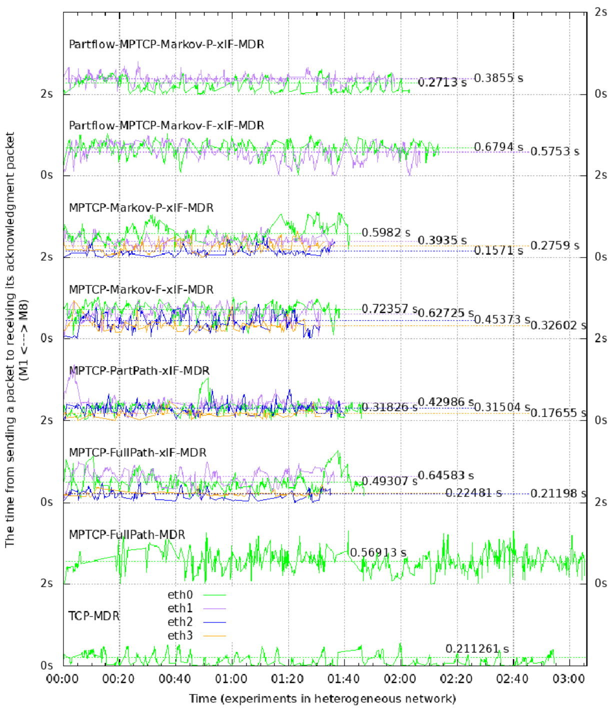


FIGURE 26. The end-to-end delay of each protocol in heterogeneous scenarios.

different. Due to the different transmission rates of interfaces, two subflows created on eth0 and eth1 by PMMFxM and PMMPxM. PMMPxM has less delay and more stability than PMMFxM.

Because the experiments are carried out in FEP [53], the test results will be different from the real environment due to the limitation and influence of hardware conditions. However, the feasibility or correctness of the proposed framework, protocols and algorithms are verified by the experiments in this paper. Our next work will test and improve the framework, protocols and algorithms in a real environment to enhance practicability.

6) DISCUSSION

In this paper, the work presented attempts to evaluate whether the improved MPTCP and the improved MDR can help latency-sensitive applications to achieve high QoE. To evaluate the performance in the homogeneous and heterogeneous scenarios, we use *iperf* and *nc* as measurement tools to generate real-time data traffic and record the throughput and RTTs of the paths in the testing process. For the homogeneous/heterogeneous cases, ten combinations of protocols (TM, MFM, MFxM, MPxM, MMFxM, MMPxM, PMMFxM, PMMPxM, PMMFxM100, PMMFxM200 and PMMFxM300) were tested to evaluate the capability of MPTCP to carry latency-sensitive application traffic. For each experiment scenario, we considered the average delay and throughput and compared among them.

Because the improved MPTCP and the improved MDR can complement each other, the improved MDR can improve the utilization of multiple paths, and the improved MPTCP can implement robustness. Although the improved MPTCP with the improved MDR can marginally underutilize multiple paths in some cases, it can reduce opportunistic retransmission, which can reduce the extra occupation of network resources. Specifically, MMPxM had an obvious effect in this respect, as shown in Figs. 13 and 17, and the results show that MMPxM performed more efficient packet transmission regardless of the performance differences of multiple paths. It is worth noting that in all symmetric/asymmetric scenarios, MFxM, MPxM, MMFxM, MMPxM, PMMFxM, PMMPxM, PMMFxM100, PMMFxM200 and PMMFxM300 enable a significant throughput increase compared to TM.

In the homogeneous and heterogeneous scenarios used in this paper, for the receive buffer size, 6*RB is the key value to make good use of multipath. When the receive buffer is less than 6*RB, there will be a serious head-of-line blocking. For the moment, the value of the receive buffer size is hard-coded in the Linux kernel for each test; in our further work, the value will be automatically adapted.

Multipath routing is discussed in Section III.D. In the experiments in Section V, the following main OSPF-MDR configuration parameters are used: 2HopRefresh = 3, HelloInterval = 1 and LSAFullness = 3. These parameters affect the overhead of maintaining routing updates and its efficiency. OSPF-MDR allows routers to originate both full-topology LSAs and partial-topology LSAs. In a dense network, partial-topology LSAs are typically much smaller than full-topology LSAs, thus achieving better scalability. The value of LSAFullness can be 0, 1, 2, 3 or 4. If LSAFullness = 0, an LSA includes only a minimum set of neighbors. This choice results in the minimum amount of LSA flooding overhead, but does not ensure routing along shortest paths. If LSAFullness = 1, the router originates a min-cost LSAs, which provide routing along shortest paths. Setting LSAFullness to 2 also provides shortest-path routing, but allows the router to advertise additional neighbors to provide redundant routes. If LSAFullness = 4, the router originates full-topology LSAs, which include all routable and full neighbors. If LSAFullness = 3, the router originates MDR full LSAs, which cause each MDR to originate a full-topology LSA while other routers originate minimal LSAs. This choice provides routing along nearly shortest paths with relatively low overhead. OSPF-MDR allows the use of differential Hellos and full Hellos. Full Hellos are sent every 2HopRefresh Hellos, and differential Hellos are sent at all other times (every HelloInterval seconds). If 2HopRefresh = 3, every third Hello is a full Hello. If 2HopRefresh = 1, then only full Hellos are to send. Differential Hellos are used to reduce overhead and to allow Hellos to be sent more frequently, for faster reaction to topology changes. In our next phase of work, the overhead of maintaining routing updates and its efficiency will be assessed in detail.

The evaluation was performed by Android virtual machines in VirtualBox, so seemingly there might be no significant difference with Linux virtual machines. One advantage of this is that the modified code on the simulation platform can be almost directly used in real mobile operating systems [53]. We created mobile device nodes to communicate between nodes using WIFI for delivering data in the NS-3 simulator. One of the main objectives was to test the feasibility of the proposed QoE-driven MPTCP-based data delivery model; therefore, we simplified the experiment by fixing the location of nodes. The model will be tested in complex scenarios in subsequent work. In addition, HMM was used in the connection states evaluation process based on the different transmission rates of interfaces. Because the experiments were carried out in FEP [53], it is not mentioned how well it scales for different network sizes, how it affects the energy consumption on nodes, or how much computational power is needed. These studies will be performed in our future work.

VI. CONCLUSION

In this paper, to achieve reliable and efficient multipath data transmission in MANETs, we provided and investigated a QoE-driven MPTCP-based data delivery model in MANETs. We presented hidden Markov model-based optimal-start multipath routing that can effectively predict a mobile node's near future network connection state according to its past connection state. We studied and improved the algorithms of both multihop routing and establishing subpaths in MANETs. The test results show that our algorithms can offer more efficient use of multiple subpaths and better network traffic load balancing. The feasibility and effectiveness of the data delivery model are verified; however, we need to do further work to achieve a better application-level QoE.

APPENDIXES

APPENDIX A
MULTIPATH ROUTING ALGORITHM

Algorithm for HMM-Based Optimal-Start Multipath Routing

```
 1: VAR: array_nexthop = [(nexthop1, new1, con1),
 2:         (nexthop2, new2, con2), . . ., (nexthopN, newN, conN)]
 3:     //new=0, the nexthop is newly connecting with the node,
 4:     //    and the item is newly inserted into array_nexthop.
 5:     //new=1, the nexthop is already connecting with the node.
 6:     //con=0, the connection state is connect.
 7:     //con=1, the connection state is disconnect.
 8: PROCEDURE: find_next_hops
 9:     // The function is used to periodically find the latest
10:     // routing entries, the periodic interval is 2 seconds
11:     use OSPF-MDR to find nexthops
12:     for each nexthop in nexthops do
13:         if nexthop exists in array_nexthop then
14:             update array_nexthop, set new=1 of nexthop
15:         else
16:             insert (nexthop, new=0, con=0) into array_nexthop
17:         end if
18:     end for
19:     insert next-hops into array_nexthop
20: END PROCEDURE
21: PROCEDURE: hmm_predict
22:     // the Viterbi algorithm is used to predict
23:     // the connection state of the next-hops after 2 seconds
24:     VAR: obs_seq      // the sequence of observations
25:     VAR: hid_seq      // the most likely hidden state sequence
26:     VAR: start_p      // the start probability array
27:     VAR: tran_p       // the transition probability matrix
28:     VAR: emis_p       // the emission probability matrix
29:     VAR: delta        // the local probability matrix
30:     VAR: psi          // store max index matrix
31:     VAR: K            // the number of hidden states
32:     VAR: T     // the length of the observation sequence
33:     for i in range [0, K) do
34:         delta[0][i] = start_p[i] * emis_p[obs_seq[0]][i];
35:         psi[0][i] = 0;
36:     end for
37:     for j in range [1, T) do
38:         for i in range [0, K) do
39:             //find maximum value and index
40:             find_max_value_index(delta[j-1], tran_p[i],
41:             emis_p[obs_seq[j]][i], K, delta[j][i], psi[j][i]);
42:         end for
43:     end for
44:     hid_seq[T-1] = find_max_index(delta[T-1], K);
45:     for t in range [T-2, -1) do
46:         hid_seq[t] = psi[t][hid_seq[t+1]];
47:     end for
48:     update array_nexthop, set values of new and con
49: END PROCEDURE
50: PROCEDURE: update_routing_table
51:     // used to update routing table according to array_nexthop
52:     for (nexthop, con) in enumerate(array_nexthop) do
53:         if con == 0 then
54:             insert nexthop into routing table
55:         end if
56:     end for
57: END PROCEDURE
58: PROCEDURE: main
59:     INPUT: obs_seq          // e.g. 2 1 0
60:     VAR: hid_seq = ('connect': 0, 'disconnect': 1)
61:     VAR: start_p = {'connect': 0.65, 'disconnect': 0.35}
62:     VAR: tran_p = {
63:         'connect': {'connect': 0.7, 'disconnect': 0.3},
64:         'disconnect':{'connect': 0.4, 'disconnect': 0.6},
65:     }
66:     VAR: emis_p = {
67:         'connect': {'good': 0.55, 'moderate': 0.35, 'bad': 0.1},
68:         'disconnect': {'good': 0.1, 'moderate': 0.3, 'bad': 0.6},
69:     }
70:     while true do
71:         find_next_hops()
72:         hmm_predict(obs_seq, hid_seq, start_p, tran_p, emis_p)
73:         update_routing_table(hid_seq)
74:         sleep 2 s            // or sleep 1 s
75:     done
76: END PROCEDURE
```

APPENDIX B
QoE-DRIVEN PACKET SCHEDULING ALGORITHM

Algorithm for QoE-Driven Packet Scheduling

```
VAR: if_quality[i]    //quality of if_i, 0≤if_quality[i]≤10000
                      // initial value of if_quality[i] is 10000
VAR: if_rtt[i]        // RTT for subflow on if_i
VAR: if_loss[i]       // packet loss rate on if_i
VAR: if_metric[i]     // metric of if_i according to destination IP
1: PROCEDURE: update_if_quality
2:     // If the fast retransmit algorithm detectes the loss,
3:     // packet is only re-transmitted over the same subflow.
4:     if loss on if_i then
5:         if if_quality[i] ≥ 100 then if_quality[i] − = 100
6:     end if
```

```
 7:    // If the loss is detected by an expiration of the RTO timer,
 8:    // the packet can be re-transmitted over both
 9:    // the same subflow and an additional subflow.
10:    if RTO on if_i then
11:        if if_quality[i] $\geq$ 1000 then if_quality[i] $-=$ 1000
12:    end if
13:    // If received an ACK from network interface if_i
14:    if ACK on if_i then
15:        if if_quality[i] $\leq$ 9999 then if_quality[i] $+=$ 1
16:    end if
17: END PROCEDURE
18: PROCEDURE: get_if_metric
19:    INPUT: d_ip                    // destination IP address
20:    if_metric[i] = -1              // initialize if_metric[]
21:    for each destination in RIB    // RIB: kernel routing table
22:        if destination == d_ip then
23:            read if_i & metric
24:            update if_metric[i] = metric
25:        end if
26:    end for
27: END PROCEDURE
28: PROCEDURE: send_packet
29:    INPUT: priority            // the priority of APP or socket
30:    INPUT: if_quality[i]
31:    VAR: NDS                   // normal delivery speed
32:    // set delivery speed based on if_quality[i] & priority
33:    if 8000 < if_quality[i] $\leq$ 10000 then      //good
34:        if priority == 1,2,3 then NDS
35:        if priority == 4 then (NDS - NDS$\gg$5)
36:        if priority == 5 then (NDS - NDS$\gg$4)
37:        if priority == 6 then (NDS - NDS$\gg$3)
38:    end if
39:    if 5000 < if_quality[i] $\leq$ 8000 then       //moderate
40:        if priority == 1 then NDS
41:        if priority == 2 then (NDS - NDS$\gg$6)
42:        if priority == 3 then (NDS - NDS$\gg$5)
43:        if priority == 4 then (NDS - NDS$\gg$4)
44:        if priority == 5 then (NDS - NDS$\gg$3)
45:        if priority == 6 then (NDS - NDS$\gg$2)
46:    end if
47:    if if_quality[i] $\leq$ 5000 then              //bad
48:        if priority == 1 then NDS
49:        if priority == 2 then (NDS - NDS$\gg$1)
50:        if priority == 3 then (NDS - NDS$\gg$1 - NDS$\gg$2)
51:        if priority == 4,5,6 then (NDS - NDS$\gg$1 - NDS$\gg$2 - NDS$\gg$3)
52:    end if
53: END PROCEDURE
54: PROCEDURE: main
55:    INPUT: priority            // APP priority $\in$ [1, 6]
56:    INPUT: if_quality[i]
57:    VAR: d_ip          // destination IP address from packet
58:    create_thread(update_if_quality);
59:    get_if_metric(d_ip);
60: get index of if_metric where if_metric[i]!=-1,assume: l,m,n
61:    if if_rtt[l] < if_rtt[m] < if_rtt[n] then
62:        send_packet(priority, if_quality[l]);
63:    end if
64:    if if_rtt[l] = if_rtt[m] = if_rtt[n] then
65:        if if_loss[l] < if_loss[m] < if_loss[n] then
66:            send_packet(priority, if_quality[l]);
67:        end if
68:    end if
69: END PROCEDURE
```

APPENDIX C
THE IMPROVED MULTIHOP ROUTING ALGORITHM

The improved multihop routing algorithm is detailed below.

(1) If the number of next hops from M1 to M8 is 3, then add 3 routing table entries, as represented in equation 4.

(2) If the number of next hops from M1 to M8 is 2, then add the following 3 routing table entries.

nexthop via nexthop-ip-1 eth0 to 112.26.1.8
nexthop via nexthop-ip-2 eth0 to 112.26.2.8
nexthop via nexthop-ip-1 eth0 to 112.26.3.8

(3) If the number of next hops from M1 to M8 is 1, then add the following 3 routing table entries.

nexthop via nexthop-ip-1 eth0 to 112.26.1.8
nexthop via nexthop-ip-1 eth0 to 112.26.2.8
nexthop via nexthop-ip-1 eth0 to 112.26.3.8

Algorithm for Improving Multihop Routing Protocol

```
VAR: num_nexthop        // the number of next hops
VAR: no_if      // network interface number corresponding to
                // network interface (e.g., e1,e2,e3)
VAR: rib              // route information base
 1: BEGIN:
 2:    PROCEDURE: add_routing_entry
 3:    // here, we assume that the destination ip is 112.26.0.5
 5:        if p0 = 112.26.0.5 then
 6:            p1 = 112.26.1.5
 7:            p2 = 112.26.2.5
 8:            p3 = 112.26.3.5
 9:        end if
10:        num_nexthop = 0
11:        nexthop = rib->nexthop
12:        while nexthop != NULL do
13:            nexthop = nexthop->next
14:            num_nexthop++;
15:        done
16:        add_routing_entry (p1, rib, num_nexthop, 1);
17:        add_routing_entry (p2, rib, num_nexthop, 2);
18:        add_routing_entry (p3, rib, num_nexthop, 3);
19:        return add_routing_entry (p0, rib, 0, 0);
20:    END PROCEDURE
22:    PROCEDURE: choose_nexthop
23:        if (num_nexthop == 1
24:             || (num_nexthop == 2 && (no_if == 1 || no_if == 3))
25:             || (num_nexthop == 3 && no_if == 1)) **then**
26:             use the first hop
27:         **end if**
28:         **if** ((num_nexthop == 2 && no_if == 2)
29:             || (num_nexthop == 3 && no_if == 2)) **then**
30:             use the second hop
31:         **end if**
32:         **if** (num_nexthop == 3 && no_if == 3) **then**
33:             use the third hop
34:         **end if**
35:     **END PROCEDURE**
36. **END**
```

REFERENCES

[1] K. Yedugundla, S. Ferlin, T. Dreibholz, Ö. Alay, N. Kuhn, P. Hurtig, and A. Brunstrom, "Is multi-path transport suitable for latency sensitive traffic?" *Comput. Netw.*, vol. 105, pp. 1–21, Aug. 2016.
[2] A. Nikravesh and Y. H. Guo, "An in-depth understanding of MPTCP on mobile devices: Measurement and system design," in *Proc. ACM MobiCom*, 2016, pp. 189–201.
[3] A. Ford and C. Raiciu, *TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 6824, 2013.
[4] (2019). *TCP Extensions for Multipath Operation With Multiple Addresses*. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-mptcp-rfc6824bis/
[5] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, "A survey on recent advances in transport layer protocols," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3584–3608, Aug. 2019.
[6] P. Goudarzi and M. Hosseinpour, "QoE enhancement for video transmission over MANETs using distortion minimization," *Scientia Iranica*, vol. 19, no. 3, pp. 696–706, Jun. 2012.
[7] C. Gottron, A. Konig, M. Hollick, S. Bergstrasser, T. Hildebrandt, and R. Steinmetz, "Quality of experience of voice communication in large-scale mobile ad hoc networks," in *Proc. 2nd IFIP Wireless Days (WD)*, Dec. 2009, pp. 1–6.
[8] (2017). *iOS: MPTCP Support in iOS 7*. [Online]. Available: https://support.apple.com/en-us/HT201373
[9] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "Poster: Evaluating android applications with multipath TCP," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2015, pp. 230–232.
[10] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 465–478, Apr. 2015.
[11] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cui, and A. Ylä-Jääski, "Tolerating path heterogeneity in multipath TCP with bounded receive buffers," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 1–14, Jun. 2013.
[12] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proc. ACM SIGCOMM Workshop Capacity Sharing Workshop (CSWS)*, Chicago, IL, USA, 2014, pp. 27–32.
[13] D. Zhou, W. Song, and M. Shi, "Goodput improvement for multipath TCP by congestion window adaptation in multi-radio devices," in *Proc. IEEE 10th Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2013, pp. 508–514.
[14] S. Ferlin-Oliveira, T. Dreibholz, and O. Alay, "Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks," in *Proc. IEEE 22nd Int. Symp. Qual. Service (IWQoS)*, Hong Kong, May 2014, pp. 123–128.
[15] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1222–1227.
[16] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 266, Oct. 2011.
[17] A. A. Barakabitze, I.-H. Mkwawa, L. Sun, and E. Ifeachor, "QualitySDN: Improving video quality using MPTCP and segment routing in SDN/NFV," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Montreal, QC, USA, Jun. 2018, pp. 182–186.
[18] H. K. Yarnagula, R. Anandi, and V. Tamarapalli, "Objective QoE assessment of dash adaptation algorithms over multipath TCP," in *Proc. 11th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2019. Bengaluru, India, 2019, pp. 461–464.
[19] S. K. Park, A. Bhattacharya, M. Dasari, and S. R. Das, "Understanding user perceived video quality using multipath TCP over wireless network," in *Proc. IEEE 39th Sarnoff Symp.*, Newark, NJ, USA, Sep. 2018, pp. 1–6.
[20] A. Elgabli and V. Aggarwal, "SmartStreamer: Preference-aware multipath video streaming over MPTCP," *IEEE Trans. Veh. Technol.*, vol. 68, no. 7, pp. 6975–6984, Jul. 2019.
[21] A. Elgabli, K. Liu, and V. Aggarwal, "Optimized preference-aware multipath video streaming with scalable video coding," *IEEE Trans. Mobile Comput.*, to be published.
[22] S. Afzal, V. Testoni, J. F. F. de Oliveira, C. E. Rothenberg, P. Kolan, and I. Bouazizif, "A novel scheduling strategy for MMT-based multipath video streaming," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, 2018, pp. 206–212.
[23] A. A. Barakabitze, L. Sun, I.-H. Mkwawa, and E. Ifeachor, "A novel QoE-centric SDN-based multipath routing approach for multimedia services over 5G networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–7.
[24] E. B. Smida, S. G. Fantar, and H. Youssef, "Video streaming forwarding in a smart city's VANET," in *Proc. IEEE 11th Conf. Service-Oriented Comput. Appl. (SOCA)*, Paris, France, Nov. 2018, pp. 1–8.
[25] J. Vlaovic, M. Vranjes, D. Grabic, and D. Samardzija, "Comparison of objective video quality assessment methods on videos with different spatial resolutions," in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Osijek, Croatia, Jun. 2019, pp. 287–292.
[26] W. Song and D. W. Tjondronegoro, "Acceptability-based QoE models for mobile video," *IEEE Trans. Multimedia*, vol. 16, no. 3, pp. 738–750, Apr. 2014.
[27] K. Nguyen, M. G. Kibria, K. Ishizu, F. Kojima, and H. Sekiya, "An approach to reinforce multipath TCP with path-aware information," *Sensors*, vol. 19, no. 3, p. 476, Jan. 2019.
[28] S. R. Pokhrel and M. Mandjes, "Improving multipath TCP performance over WiFi and cellular networks: An analytical approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2562–2576, Nov. 2019.
[29] A. Marwa and M. Imad, "A survey of vehicular ad hoc networks routing protocols," *Int. J. Innov. Appl. Stud.*, vol. 3, no. 3, pp. 829–846, Jul. 2013.
[30] M. Karimi and D. Pan, "Challenges for quality of service in mobile ad-hoc networks," in *Proc. IEEE 10th Annu. Wireless Microw. Technol. Conf.*, Apr. 2009, pp. 1–5.
[31] P. L. Callet, *Qualinet White Paper on Definitions of Quality of Experience (2012)*, document CH-1015, European Network on Quality of Experience in Multimedia Systems and Services, Lausanne, Switzerland, Version 1.2, Mar. 2013.
[32] L. Yao, J. Wang, X. Wang, A. Chen, and Y. Wang, "V2X routing in a VANET based on the hidden Markov model," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 3, pp. 889–899, Mar. 2018.
[33] M. Tajima, "An innovations approach to viterbi decoding of convolutional codes," *IEEE Trans. Inf. Theory*, vol. 65, no. 5, pp. 2704–2722, May 2019.
[34] E. Theodosis and P. Maragos, "Analysis of the Viterbi algorithm using tropical algebra and geometry," in *Proc. IEEE 19th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Kalamata, India, Jun. 2018, pp. 1–5.
[35] R. A. Rashid, H. Harun, Z. Mansor, N. Shamsudin, and S. M. Nor, "Pruning the algorithm complexity of the add-compare select unit (ACSU) for the Viterbi decoder—A review," in *Proc. IEEE 5th Int. Conf. Smart Instrum., Meas. Appl. (ICSIMA)*, Songkla, Thailand, Nov. 2018, pp. 1–5.
[36] S. Ahmed, F. Siddique, M. Waqas, M. Hasan, and S. U. Rehman, "Viterbi algorithm performance analysis for different constraint length," in *Proc. 16th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Islamabad, Pakistan, Jan. 2019, pp. 930–932.
[37] *Mobile Ad Hoc Network (MANET) Extension of OSPF*. [Online]. Available: https://tools.ietf.org/html/rfc5614
[38] *OSPF for IPv6*. [Online]. Available: https://tools.ietf.org/html/rfc5340
[39] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.

[40] J.-P. Sheu, L.-W. Liu, R. Jagadeesha, and Y.-C. Chang, "An efficient multipath routing algorithm for multipath TCP in software-defined networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Athens, Greece, Jun. 2016, pp. 371–376.

[41] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Appl. Math.*, vol. 26, no. 1, pp. 105–115, Jan. 1990.

[42] F. Zhou, T. Dreibholz, X. Zhou, F. Fu, Y. Tan, and Q. Gan, "The performance impact of buffer sizes for multi-path TCP in Internet setups," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Taipei, Taiwan, Mar. 2017, pp. 9–16.

[43] C. Paasch and S. Barrãé, *Multipath TCP in the Linux Kernel*. [Online]. Available: http://www.multipath-tcp.org

[44] S. Barrãé, C. Paasch, and O. Bonaventure, "MultiPath TCP: From theory to practice," in *Proc. Int. Conf. Res. Netw.*, 2011, pp. 444–457.

[45] *Extensions for Network-Assisted MPTCP Deployment Models*. document Internet draft, draft-boucadair-mptcp-plain-mode-10, Work in Progress, Mar. 2017.

[46] *Use Cases and Operational Experience with Multipath TCP*, document IETF RFC 8041, Jan. 2017.

[47] C. Raiciu, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. 9th USENIX Conf. Networked Syst. Design Implement.*, San Jose, CA, USA, Apr. 2012, pp. 1–14.

[48] F. Yang and P. Amer, "Non-renegable selective acknowledgments (NR-SACKs) for MPTCP," in *Proc. 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Mar. 2013, pp. 1113–1118.

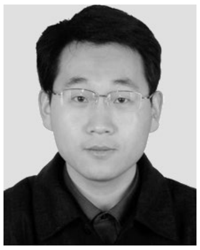
[49] H. Adhari, T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "Evaluation of concurrent multipath transfer over dissimilar paths," in *Proc. IEEE Workshops Int. Conf. Adv. Inf. Netw. Appl.*, Singapore, Mar. 2011, pp. 708–714.

[50] T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tuxen, "On the use of concurrent multipath transfer over asymmetric paths," in *Proc. IEEE Global Telecommun. Conf. GLOBECOM*, Miami, FL, USA, Dec. 2010, pp. 1–6.

[51] (2019). *Multipath-Routing and MPTCP-Based Data Delivery Over MANETs*. [Online]. Available: https://github.com/ztguang/DoM

[52] (2019). *OSPF MANET Designated Routers (OSPF-MDR) Implementation*. [Online]. Available: https://www.nrl.navy.mil/itd/ncs/products/ospf-MANET

[53] T. Zhang, S. Zhao, B. Cheng, B. Ren, and J. Chen, "FEP: High fidelity experiment platform for mobile networks," *IEEE Access*, vol. 6, pp. 3858–3871, 2018.



TONGGUANG ZHANG received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, in June 2018. He is currently an Associate Professor of computer science with Xinxiang University. His current research interests include the mobile Internet technology, the Internet of Things technology, communication software and distribute computing, embedded systems, and service computing.

SHUAI ZHAO received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, in June 2014. He is currently an Associate Professor of computer science with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include the Internet of Things technology and service computing.

BO CHENG (Member, IEEE) received the Ph.D. degree in computer science from the University of Electronics Science and Technology of China, Chengdu, China, in 2006. He is currently a Professor of computer science with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include network services and intelligence, the Internet of Things technology, communication software, and distribute computing.