Experimental Study of Disruption-Tolerant Transport Protocol for Mobile Ad-Hoc Networks with Connection Diversity

Xiaoshan Chang, Zenghua Zhao, Bingxue Diao, Tao Li
Tianjin Key Laboratory of Advanced Networking (TANK), School of Computer Science and Technology
Tianjin University, Tianjin, China 300354
Email: zenghua@tju.edu.cn

***Abstract*—Due to node mobility, the network topology of a mobile ad-hoc network (MANET) usually varies from full connection to intermittent connection, showing a connection diversity. There have been many well-known routing protocols for full connection and intermittent connection situations respectively. However, it is hard to find one-fit-to-all routing protocol suitable in all connection situations. In this paper, we propose a unified transport framework for MANET with connection diversity, named by DTTP (Delay Tolerant Transport Protocol). DTTP provides a scheme to integrate routing protocols in both paradigms. It is only implemented in end nodes (sources and destinations) and the intermediate nodes remain untouched. The decision on which protocol to use for transmitting a given message from source to destination is made on application level and transparent to applications. To this end, DTTP defines an unified API for application development, and designs a heartbeat mechanism to determine the network connection situations. We implement DTTP in our testbed and evaluate its performance. The experiment results verify its fundamental functionalities.**

I. Introduction

Mobile ad-hoc network (MANET) have been attracted many research activities over the last decades, motived by the current or foreseeable applications such as vehicular ad hoc networks, disaster relief and wildlife monitoring. Due to the mobility of nodes, the network topology of the MANET usually varies from full connection to intermittent connection both in temporal and spatial spaces, showing a connection diversity characteristic [1].

The conventional routing protocols designed for MANET focus on fully connected situations, aiming at establishing an end-to-end path between the source and the destination [2]–[5]. Whereas, routing protocols specified for the intermittently connected MANET assume that most of the time there does not exist a complete path from a source to a destination, thus adopt store-carry-forward paradigm [6]–[9]. From this perspective, the intermittently connected MANET belongs to the general category of Delay Tolerant Networks (DTN) [10], [11]. However, it is hard to find a one-fit-to-all routing protocol suitable in all connection situations, since the routing protocol appropriate in fully connected situations performs poorly or even does not work under intermittent connectivity without end-to-end paths, and vice versa [9].

It is beneficial to combine routing protocols in fully connected and intermittently connected MANET against the connectivity variety [12]–[16]. HYMAD [15] partitions the nodes into groups according to the topology connectivity. Inside a group, DSDV [2] is deployed to achieve high performance, whereas, Spray-and-Wait [8] is used to transmit messages between groups. Lakkakorpi *et al.* [12], [14] propose an adaptive routing to transmit messages from source to destination using either AODV or DTN routing, depending on current node density, message size and path length to destination. Liu *et al.* [13] switch between AODV [4] and Spray-an-Wait from the bandwidth allocation perspective. However, they all focus on how to coordinate the two routing components in two connectivity extreme ends in order to achieve better network performance, and do not have a unified API for applications. Moreover, previous works are evaluated in simulations and more or less do not consider the implementation issues in practice.

In this paper, we propose a general framework for the integration of the routing protocols both in fully connected and intermittently connected MANET. Particularly, we design DTTP, a delay tolerant transport protocol for MANET with connection diversity. Different from pervious works [12]–[16], DTTP considers both TCP/IP protocol stack and DTN protocol stack, and defines an unified API for application development. DTTP is deployed only in end nodes (sources and destinations) and the intermedia nodes remain untouched. A heartbeat mechanism is deployed to determine the end-to-end connectivity and thus the protocol stacks (or, the routing protocols) are switched accordingly. The switch is transparent to applications thanks to the unified API.

In summary, the contributions of the paper are two-folds:

(1) We propose DTTP, a general transport framework for integrating routing protocols in both connectivity extreme ends to tackle the connection diversity in MANET. DTTP is designed to support both DTN protocol stack and TCP/IP protocol stack. The switch between the two protocol stacks (*i.e.*, routing protocols) are adaptive to connectivity situations and transparent for applications.

(2) We implement DTTP in our testbed and verify its fundamental functionalities.

The remainder of this paper is organized as follows. In Section II, we briefly introduce the background of MANET and DTN. In Section III, we describe the motivation of

2324-9013/16 $31.00 © 2016 IEEE
DOI 10.1109/TrustCom.2016.247

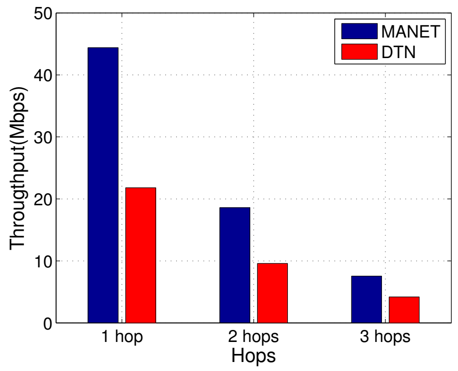


Fig. 1. Throughput of MANET and DTN in well-connected situations

designing DTTP. Section IV describes the design of DTTP. Section V describes the implementation details. In Section VI experimental results are presented. Finally, Section VII concludes this paper.

II. Background and Related Work

A. Mobile ad-hoc networking and routing protocols

Mobile ad-hoc networks were first studied under the assumptions of moderate node mobility and sufficient density to ensure end-to-end connectivity. Therefore, conventional routing protocols for MANET are designed to establish an end-to-end path for the source and the destination, which have been well studied. Conventional MANET routing protocols can be roughly divided into proactive (*e.g.*, DSDV [2], OLSR [3]), reactive (*e.g.*, AODV [4]), and hierarchical (*e.g.*, ZRP [5]) .

B. Delay-tolerant Networking and the Bundle Protocol

When the density of nodes diminishes or the mobility of nodes enhances in MANET, end-to-end connectivity might disappear. In the extreme environments, the network will intermittently connect, and fall into the category of delay-tolerent networks [8].

In order to solve the interoperability between heterogeneous challenged networks, DTN architecture introduces the Bundle layer between the transport layer and the application layer [11], [17]. Bundle layer, as an end-to-end Message-Oriented convergence layer, employs store and forward message exchange mechanism, hop-by-hop message confirmation mechanism and selected end-to-end acknowledge mechanism, providing persistent storage. In addition, the custody transfer (CT) mechanism of DTN inter-nodes message retransmission and confirmation increases the reliability of message transmission. In summary, DTN is a message-based overlay network architecture. DTN endpoints are identified by endpoint identifiers (EID), which is specified in a URL-style format: *scheme: specific address*.

Routing protocols for intermittently connected MANET (or DTN, hereafter) have been studied as well. DTN routing often relies on information replication for forwarding bundles to maximize delivery probability and/or minimize transmission time [6]–[9].

C. Related work

Since conventional MANET routing protocols are designed for well-connected network, they perform poorly or even cannot work under intermittently connectivity. On the other hand, the DTN routing protocols under well connectivity deteriorate the network performance too for its redundant message replication. Therefore there have been attracted research on integrating both kinds of the routing protocols in extreme connectivity ends [12]–[16].

HYMAD [15] is a hybrid DTN-MANET routing protocol. In HYMAD, nodes identify groups of connected neighbors. It forwards messages using MANET routing within the groups, while inter-group communication occurs using the DTN paradigm. [12] augments AODV route discovery to become an implicit service discovery mechanism for DTN routers. AODV is used as a vehicle to locate DTN routers as possible optimization for communication or fallback in case no end-to-end path can be determined. CAR [18] is able to deliver messages synchronously (*i.e.*, without storing them in buffers of intermediate nodes when there are no network partitions between the sender and the receiver) and asynchronously (*i.e.*, by means of a store-and-forward mechanism when there are partitions). The delivery process depends on whether or not the recipient is present in the same connected region of the network as the sender.

A hybrid algorithm of MANET and DTN is also proposed in [16]. The authors assume that all nodes choose the MANET or DTN network model, each node can dynamically select the most appropriate network model according to their own parameters and the surrounding environment. The selection algorithm proposed by the authors is based on three parameters: battery remaining power, running speed and acceleration, the number of neighbour nodes. Some nodes run a DTN routing protocol and others run MANET routing protocol. [13] proposes an adaptive routing protocol which allocates bandwidth between its multi-hop forwarding component and its mobility-assisted routing component dynamically to improve bandwidth utility.

Different from these works, DTTP proposes a general transport framework for integration of the routing protocols in both extreme connectivity ends.

III. Motivation

As the wireless network topology varies, users likely encounter environment with varying connectivity where the underlying topology sometimes resembles a MANET and other times a DTN. A robust protocol can adapt to intermittently connected networks and well-connected networks. Therefore we start our work with the following questions: Can we design a transport protocol that works robustly both in MANET with connection diversity? Can this protocol be transparent to users? In this section, we will explain why we design such a protocol through a case of experiment.

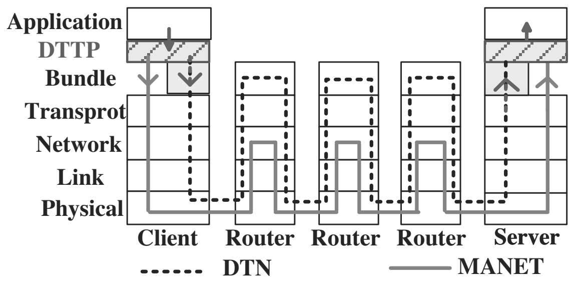


Fig. 2. DTTP in protocol stack

The state-of-the-art wireless routing protocols design for their specific environment. Traditional link-state or distance vector MANET protocols break down in intermittently connected scenarios, resulting in a deluge of topology updates for the proactive routing protocols and route error and new route requests messages for the reactive routing protocols. The reason is that MANET protocol assumes the availability of the contemporaneous end-to-end path. Traditional MANET lacks the specialized solution to deal with the disconnection of the wireless network in extreme circumstance, therefore when the connection is disrupted, the network performance will decrease significantly, even cause network not to run. In contrast, DTN routing protocols can deal with intermittently connected scenarios through the way of store-carry-forward, but suffer poor performance in well-connected environment.

We conduct an experiment in our testbed to show the performance of both kinds of routing protocols in well-connected scenario. We use UBNT RouterStationPro, an embedded open source wireless router, as our wireless node in our testbed, the details of RouterStationPro are listed in Table I. We deploy IBR-DTN [19] in our wireless nodes. In this experiment we adopt OLSR [3] as MANET routing protocol and PROPHET [7] as DTN protocol. We also measured other routing protocols and obtained the similar results. We measure the end-to-end UDP throughput under 1 hop, 2 hops and 3 hops scenarios respectively.

Fig. 1 shows that the throughput of OLSR (MANET) outperforms PROPHET (DTN) by 2 times under any scenario in well connectivity. The reason is that DTN routing protocol usually use packet replication to reduce delay, however packet replication consumes more network resource and yields little benefits in well-connected networks. Moreover, in DTN architecture the bundle layer is inserted on top of the transport layer, which results in longer header of the packet and more process. The experiment results are also confirmed in [1], [8].

In addition, DTN transmits messages in asynchronous hop-by-hop mechanism to avoid the need of an end-to-end path. However, these usually compel the user/application to obey asynchronous fashion and require the application designed on top of the bundle layer. For many Internet applications, we have to notice that the user prefer to use Socket programming interface and keep the end-to-end semantics (.e.g transparent transmission). Therefore, how to make the routing integration transparent to applications is important for application development in practice.

IV. Design of DTTP

In this section, we overview the design of DTTP, a delay tolerant transport protocol that provides a general framework for integrating routing protocols in fully connected and intermittently connected MANET.

A. Design Principles

We adhere to the following design principles in order to make DTTP adaptive to the diverse connectivity and provide a unified API to shield the lower different protocol interfaces.

1) Sensitive to the connectivity. DTTP should be sensitive to the dynamic connectivity of the network, and is able to detect the connection status (well-connected or intermittently-connected) correctly in real time.

2) Easy to switch routing protocols. DTTP should be able to switch to the appropriate routing protocol for well-connected or intermittently connected MANET according to the current connection status.

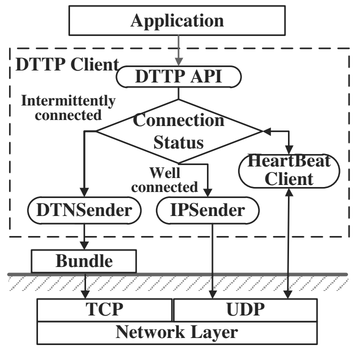
3) Easy to deploy. DTTP should be easy to deploy in practice. It should not need too much modification on current protocol stacks.

4) Transparent to the upper layer applications. DTTP should be transparent to the upper layer applications. In other words, the switch to different routing protocols or different protocol stacks should be shielded to the application development. Since conventional MANET adopts TCP/IP protocol stack, TCP/UDP socket is the default application interface. Whereas, DTN uses Bundle layer, which provides a different application interface. DTTP should be able to provide an unified application interface for the user development.

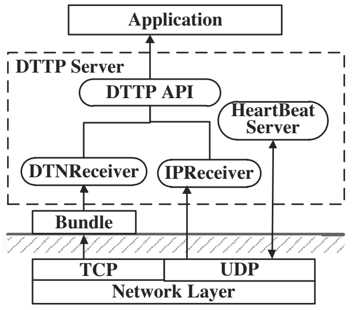
B. DTTP Overview

DTTP provides a general transport framework for integrating routing protocols in fully-connected and intermittently-connected scenarios against the connection diversity in MANET. The position of DTTP in the protocol stack is shown in Fig. 2. It says that DTTP only works in end nodes (*i.e.* sources and destinations), and the intermediate nodes remain untouched. This makes it easy to deploy. DTTP lies on top of the Bundle layer and the transport layer, thus it can define a unified interface for the upper layer applications. The TCP/IP protocol stack and Bundle Protocol stack work independently in all the nodes. Since the protocol stack usage decision is made in the source node by DTTP, the intermediate nodes just forward the message using the corresponding protocol stack. For example, if DTTP decides to use DTN routing protocol according to current network connection status, it packetizes messages into Bundles and send them to transport layer via Bundle protocol. When the intermediate nodes receive the bundles, they will forward or buffer the bundles according to their DTN routing decision.

DTTP consists of a client and a server. As shown in Fig. 3. A DTTP client waits for receiving messages from upper



(a) DTTP client



(b) DTTP server

Fig. 3. The diagram of DTTP client and server

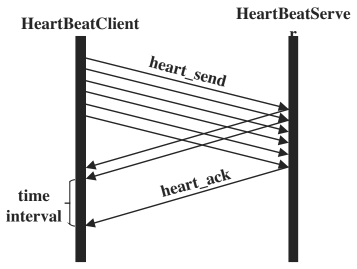


Fig. 4. The diagram of heartbeat mechanism

layer application through DTTP API. When DTTP client receives a message, it checks the current connection status. The connection status is maintained by a heartbeat mechanism. If the current connection status is well-connected, DTTP client pushes the message into TCP/UDP socket. Otherwise, DTTP client calls the DTN sender to transmit the message via Bundle protocol. The DTTP server listens for messages from the below layer. When DTTP server receives a message from the below layer, it pulls the message from bundle protocol or from TCP/UDP socket and deliver it to the upper layer application through DTTP API.

In summary, DTTP consists of three main mechanisms: 1) detecting the end-to-end connectivity, 2) switching the routing protocol, and 3) unifying application interface. We now present the details of each mechanism in the following subsections.

C. Detecting End-to-End Connectivity via Heartbeat Mechanism

To detect the end-to-end connectivity is a primary function of DTTP. How to detect the connection status quickly and correctly is challenging. We considered three approaches at the beginning of the design.

1) Utilizing the routing table

The routing table contains the routing information and maintained by the routing protocol. For source routing protocol (*e.g.*, DSR [20]), the routing table shows the routes from the source to the destination. For other routing protocol, the routing table lists the neighbor nodes information. The routing table can be used to infer the connectivity of the network. However, except source routing protocol, others only provide local connectivity information. Moreover, the routing table is maintained by routing protocol and the maintenance is either on-demand or periodically. Therefore the routing table might be stale.

2) Making use of TCP keep-alive mechanism

TCP keep-alive mechanism sends keep-alive segments at a certain time interval. If there is no response within a period of time the segment will be sent again. If the TCP connection is not alive, TCP will feedback a status to notify the application. We might make use of this notification. However, the TCP connection will be closed after the notification. That is to say, TCP connection will be closed when the network is intermittently connected. Therefore we cannot use it to determine the network connectivity.

3) Sending heartbeat packets

Fortunately, we can send heartbeat packets periodically to breakthrough the limitation of the routing table and the TCP keep-alive. The heartbeat packets are sent from the source to the destination, and the destination will acknowledge the source. The source can infer the end-to-end connectivity by the round trip time (RTT) of the heartbeat packet.

Therefore, at last we choose the heartbeat mechanism to detect the end-to-end connectivity. The heartbeat mechanism is also composed of a client (HeartBeatClient) and a server (HeartBeatServer). HeartBeatClient updates the network connection status according to the current network connectivity. To this end, it sends heartbeat packets periodically via UDP to the heartbeat server. Fig. 4 shows that the client periodically transfers heartbeat packets to the server. When the server receives a heartbeat packet, it immediately sends back an acknowledgement to the client. The client records the time interval of the two consecutive acknowledgements. If the time interval exceeds a time threshold Th_t, the connection status is intermittently connected, otherwise the connection status is well-connected. The threshold Th_t needs to be carefully selected in order to react to the network connectivity rapidly.

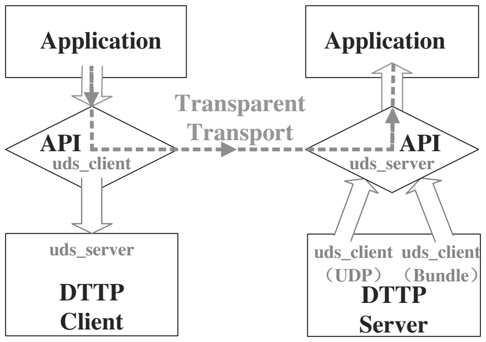


Fig. 5. DTTP API

D. Switching the Routing Protocol

DTTP switches the routing protocol according to the network connection status. Moreover, the routing protocol switch is done by switching the protocol stack actually. In this way, the routing protocol can be selected flexibly, since the routing protocol can be setup in the script and take effect after restart. If the connection status is marked as well-connected, DTTP selects TCP/IP protocol stack and use IP sender to make an end-to-end transmission via TCP socket or UDP socket. TCP/UDP socket is selected according to the application requirements. If the application requires reliable transmission, TCP socket is used, otherwise UDP socket is selected. If the connection status is intermittently-connected, DTTP switches to DTN protocol stack, the client calls DTN sender to transmit the message hop-by-hop from the source to the destination.

Since the messages might be transmitted by either DTN sender or IP sender, DTTP client needs to partition the messages and the DTTP server collects and sorts the messages before delivering them to the application.

E. Unifying Application Interface

Since DTTP needs to switch between DTN sender and IP sender according to current network connection status, it should provide an unified interface for the application shielding the API difference of the two senders. To this end, we make use of IPC (Inter Process Communication) socket. IPC socket is a Unix domain socket, which is a data communications endpoint for exchanging data between processes executing on the same host operating system. The API for IPC socket is similar to that of an Internet socket, but rather than using an underlying network protocol, all communication occurs entirely within the operating system kernel. IPC sockets use the file system as their address name space. Processes reference IPC sockets as file system *inodes*, so two processes can communicate by opening the same socket. IPC sockets support transmission of a reliable stream of bytes (SOCK_STREAM, compare to TCP), ordered an reliable transmission of datagrams (SOCK_SEQPACKET), and unordered and unreliable transmission of datagrams (SOCK_DGRAM, compare to UDP).

TABLE I
THE SETUP OF ROUTERSTATIONPRO

Items	Description
CPU	Atheros AR7161 MIPS 24K, 680MHz
MEMORY	DDR 128Mbyte
FLASH	16Mbyte
Wireless card	UBNT SR71-A
OS	OpenWrt

DTTP defines its API as in Fig. 5. From the view of the applications, they transmit data from the client (or the source) to the server (or the destination) using DTTP API just like operating via Internet socket. The source puts the data into DTTP API, and the destination gets the data out of the DTTP API. The data transmission is transparent for the applications, so is the protocol stack switch. In fact, DTTP client gets the data from the application through a IPC socket, and push the data to either the Bundle layer or the TCP/UDP socket according to the network connection status. Correspondingly, DTTP server obtains the data from the Bundle layer or the TCP/UDP socket and sorts them, then deliver them to the upper layer application.

V. IMPLEMENTATION

We implement DTTP in our testbed. In this section, we first introduce the experimental setup, then present the implementation issues of DTTP, and finally we give a case to show how to run DTTP.

A. Experimental Setup

We use a programmable platform, UBNT RouterStationPro running OpenWrt open source OS, as our hardware platform. Each of them is equipped with a UBNT SR71-A 802.11n miniPCI wireless card. We run all the experiments at 5 GHz frequency band without significant external WiFi interference. The hardware setup of RouterStationPro is listed in Table I.

DTTP can be setup with any routing protocols. Without loss of generality, we adopt OLSR (Optimized Link State Routing) [3] as the conventional MANET routing. The *olsrd* (olsr daemon) with version 0.0.6.2 [21] is deployed in our testbed. It works in the user space in the Linux operating system. OSLR is one of the most widely-used MANET routing protocols. It is an optimization of the classical link state algorithm tailored to the requirements of a MANET. The key concept used in the protocol is that of multipoint relays (MPRs). MPRs are selected nodes which forward broadcast messages during the flooding process. Link state information is generated only by nodes elected as MPRs. This technique substantially reduces the message overhead as compared to a classical flooding mechanism, where every node retransmits each message when it receives the first copy of the message. OLSR provides optimal routes (in terms of number of hops).

The standard DTN protocol is Bundle Protocol defined in RFC 5050 [17]. There are several Bundle Protocol implementations for different use cases, including DTN2 [22], ION

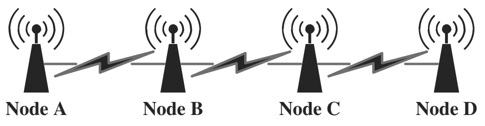


Fig. 6. Network topology

(Interplanetary Overlay Network) [23] and IBT-DTN [24]. Among them, IBR-DTN is a lightweight, modular and highly portable Bundle Protocol implementation designed for embedded systems running OpenWRT. IBR-DTN performs well, and operates very efficiently and consumes few memory resources [25], [26], therefore we chose IBR-DTN in our testbed. The DTN routing protocol is set as PROPHET [7], which is a probabilistic protocol using history of node encounters and transitivity to enhance performance.

B. Implementation Details

In this section, we describe the details of the implementation of the main components of DTTP: the heartbeat mechanism and the unified application interface.

Implementation of the heartbeat mechanism. DTTP clients runs two heartbeat threads, one takes in charge of sending heartbeat packets, the other receives the acknowledgment and sets the network connection status. The heartbeat packets are short packets and are sent to the server sequentially periodically. The time interval between two consecutive received acknowledgement packets are calculated. If the time interval is more than a threshold Th_t, then the connection status is set to intermittently-connected. The threshold Th_t depends on the round trip time of the heartbeat packets. The sending thread still sends the heartbeat packets at the DTTP client during the intermittently-connected period. While the receiving thread receives acknowledgement packets from the DTTP server for a period of time, the connection status is set to well-connected.

Implementation of the unified application interface. As shown in Fig. 5, DTTP client establishes a UDS (Unix Domain Socket) SOCK_DGRAM type of connection, which consists of a UDS client and a UDS server. The UDS client provides a classic socket interface to the application, and receives the messages from the application. UDS server transfer the messages from the application through the UDS client to the Bundle layer or the TCP/UDP socket according to the connection status.

DTTP server runs two UDS clients at the same time to receive the messages from the Bundle layer and the TCP/UDP socket respectively. The two UDS clients need to build connections with the UDS server. No matter from which UDS client the messages are received, the UDS server puts the messages in the receiving buffer sequentially and deliver them to the application. In this way, from the view of the application, DTTP provides an end-to-end logic path for their message transmission. Therefore it is transparent for the application, and the application does not need to care about the routing protocols at the lower layer.

TABLE II
THE INFORMATION OF SENDING AND RECEIVING

	node A	node D
Time period 1	"aaa" "bbb"	"aaa" "bbb"
Time period 2	"ccc" "ddd"	
Time period 3	"eee"	"ccc" "ddd" "eee"

C. A Use Case of DTTP

To use DTTP, all network nodes need to support TCP/IP stack and IBR-DTN. However they use different ways to identify nodes. The nodes are identified by IP address for TCP/IP stack, whereas the nodes are identified by EID (end-point identifiers) for IBR-DTN. To solve this, DTTP provides a *dttpsend* tool. The *dttpsend* assigns the IP address and EID to each node. The operation mode of DTTP client is *dttpsend [DTN EID] [IP]*. In addition, EchoWorker is a bundle application which is built in the daemon of IBR-DTN and relies on the *dtnd* tool [27], therefore the receiver needs to open the *dtnd* tool to receive bundles. The DTTP also provides a receiving tool of *dttprecv* to receive the messages from the *dttpsend*.

VI. EVALUATION

We evaluate DTTP in our testbed. In this section, we first present the experiment scenario and then show the experiment results.

A. Experiment Scenario

The network topology used in the experiment is depicted in Fig. 6, which consists of 4 wireless nodes (RouterStationPro). All the nodes operate in ad-hoc mode using IEEE 802.11n. We conduct the experiment in our laboratory. However, every nodes can hear each others in a small space. In order to simulate the situation that only adjacent nodes can hear each other, we use the iptables linux tool as in [28]:

#iptables -A INPUT -m mac -mac source MAC ADDR -j DROP

In this way, we can build a multi-hop network topology in the laboratory. In addition, we pull up the antenna of a node to simulate that the node moves away out of the transmission range of other nodes. For example, if we pull up the antenna of node B and node C, the link breaks down between node B and node C. In this case, there is no path from node A to node D.

In our experiment DTTP client and DTTP server operate on node A and node D respectively. This experiment scenario can provide two distinct connection features, *i.e.*, well-connected and intermittently connected. The heartbeat packets are sent to the server sequentially every 3 seconds. The time interval threshold Th_t is set to 15 seconds by experience.

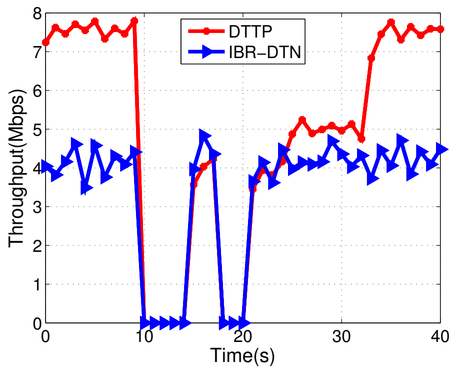


Fig. 7. Throughput of DTTP and DTN routing protocol

B. Experiment Results

We first verify the functionality of DTTP via a simple character transfer, and then measure its end-to-end throughput.

Functionality verification. We implement a sample application of sending characters to verify the primary functionalities of DTTP. The key functions are determining end-to-end connectivity status and dynamically switching to the appropriate routing protocols accordingly. To this end, we simulate a network with dynamic connectivity. The experiment experiences 3 time periods. During the first time period, the network is well-connected and shows a 3-hop linear network topology. In the second time period, we pull off the antenna of node B and node C, causing a path broken between node A and node D. Node A still have connection with node B, but node B cannot hear from node C. In the third time period, we put on the antenna of node B and node C, making the network recovering to well-connected again.

During the experiment, we record the characters sent from node A and the characters received by node D. Table II lists the results. We can see that during time period 1, node D receives what node A sends in real time. In this period, since the network is well-connected, DTTP client selects TCP/UDP socket to send the messages (we also logged this status during the experiment). The *olsrd* finds an end-to-end path between node A and node B. However, in the second time period, there is no end-to-end path available any longer. Therefore DTTP switches to use Bundle protocol, which adopts PROPHET routing protocol. node B buffers the characters sent by node A during this period, and carries them until there is a end-to-end path again in the third time period. That is why node D receives nothing during time period 2, but receives all the messages from A in time period 3.

This simple experiment verifies the primary functionalities of DTTP. It can detect the network connectivity and switch between OLSR and PROPHET routing protocols according to the network connection status. Any other routing protocols can also be adopted.

End-to-end throughput. To show the performance of DTTP, we measure its end-to-end throughput and compare it with that using IBR-DTN (PROPHET) only. In this experiment, we also simulate the dynamic network connection variety. The experiment lasts 40 seconds. During 0 $\sim$ 10s, the network is well-connected; in 10 $\sim$ 15s, the link between node B and node C is broken; in 15 $\sim$ 18s the link between node B and node C recovers; in 18 $\sim$ 21s the link between node B and node C breaks down again; in 21 $\sim$ 40s, the network is well-connected again.

Fig. 7 shows the end-to-end throughput of DTTP and IBR-DTN without DTTP during the experiments. It says that while the network is well-connected during 0 $\sim$ 10s and 21 $\sim$ 40s, the throughput of DTTP outperforms IBT-DTN by almost 2 times, consistent to what we measured in Section III. This illustrates that DTTP switches to use OLSR routing protocol when the network is well-connected. On the other hand, when the end-to-end path is unavailable during 10 $\sim$ 15s and 18 $\sim$ 21s, DTTP switches to using PROPHET, and the throughput of both DTTP and IBR-DTN decreases to zero. However, the messages sent by node A during these periods do not lose but be buffered in intermediate node B. From the throughput values at time 15s and 21s, we can see that DTTP detect the network connectivity rapidly and correctly, since the throughput increases quickly when the network recovers from intermittent connection.

In summary, the experiment results validate the primary functionalities of DTTP, and show that DTTP outperforms IBR-DTN using only one kind of routing protocol.

VII. Conclusion

In this paper, we propose DTTP, a transport framework for integrating routing protocols in both connectivity extreme ends (*i.e.*, fully-connected and intermittently-connected). DTTP is an end-to-end solution compared to other hybrid routing protocols [12]–[16]. It lies on top of Bundle layer and transport layer, and defines an unified interface for application development. DTTP adopts a heartbeat mechanism to detect the network connectivity and switches to appropriate protocol stack (thus the routing protocols) accordingly. We implement DTTP and validate its functionalities on our testbed.

DTTP is still at its early stage, we have conducted basic functional experiments with our prototype in an emulation environment, validating the principle of our approach but also encountering a few challenges. In the future work, we will enable DTTP to provide multiple APIs in different OS, including Windows. In addition,we will expand our experiment to larger network settings with mobility.

Acknowledgement

This research was supported in part by the National Natural Science Foundation of China (NSFC) under grant No. 61172063.

References

[1] X. Tie, A. Venkataramani, and A. Balasubramanian, “R3: Robust replication routing in wireless networks with diverse connectivity characteristics,” in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, MobiCom ’11, (New York, NY, USA), pp. 181–192, ACM, 2011.

[2] C. E. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers,” in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM ’94, (New York, NY, USA), pp. 234–244, ACM, 1994.

[3] P. Jacquet, “Optimized Link State Routing Protocol (OLSR).” RFC 3626, Mar. 2013.

[4] S. R. Das, C. E. Perkins, and E. M. Belding-Royer, “Ad hoc On-Demand Distance Vector (AODV) Routing.” RFC 3561, Mar. 2013.

[5] Z. J. Haas and M. R. Pearlman, “The performance of query control schemes for the zone routing protocol,” *IEEE/ACM Trans. Netw.*, vol. 9, pp. 427–438, Aug. 2001.

[6] A. Vahdat, D. Becker, *et al.*, “Epidemic routing for partially connected ad hoc networks,” 2000.

[7] A. Lindgren, A. Doria, and O. Schelén, “Probabilistic routing in intermittently connected networks,” *ACM SIGMOBILE mobile computing and communications review*, vol. 7, no. 3, pp. 19–20, 2003.

[8] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Efficient routing in intermittently connected mobile networks: the multiple-copy case,” *Networking, IEEE/ACM Transactions on*, vol. 16, no. 1, pp. 77–90, 2008.

[9] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Efficient routing in intermittently connected mobile networks: The single-copy case,” *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 1, pp. 63–76, 2008.

[10] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, “Delay-tolerant networking: an approach to interplanetary internet,” *Communications Magazine, IEEE*, vol. 41, no. 6, pp. 128–136, 2003.

[11] K. Fall, “A delay-tolerant network architecture for challenged internets,” in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM ’03, (New York, NY, USA), pp. 27–34, ACM, 2003.

[12] J. Ott, D. Kutscher, and C. Dwertmann, “Integrating dtn and manet routing,” in *Proceedings of the 2006 SIGCOMM Workshop on Challenged Networks*, CHANTS ’06, (New York, NY, USA), pp. 221–228, ACM, 2006.

[13] C. Liu and J. Wu, “Efficient adaptive routing in delay tolerant networks,” in *Communications, 2009. ICC’09. IEEE International Conference on*, pp. 1–5, IEEE, 2009.

[14] J. Lakkakorpi, M. Pitkänen, and J. Ott, “Adaptive routing in mobile opportunistic networks,” in *Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, MSWIM ’10, (New York, NY, USA), pp. 101–109, ACM, 2010.

[15] J. Whitbeck and V. Conan, “Hymad: Hybrid dtn-manet routing for dense and highly dynamic wireless networks,” *Computer Communications*, vol. 33, no. 13, pp. 1483 – 1492, 2010.

[16] Y. Kawamoto, H. Nishiyama, and N. Kato, “Toward terminal-to-terminal communication networks: A hybrid manet and dtn approach,” in *2013 IEEE 18th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 228–232, Sept 2013.

[17] K. Scott and S. C. Burleigh, “Bundle Protocol Specification.” RFC 5050, Oct. 2015.

[18] M. Musolesi and C. Mascolo, “Car: Context-aware adaptive routing for delay-tolerant mobile networks,” *Mobile Computing, IEEE Transactions on*, vol. 8, no. 2, pp. 246–260, 2009.

[19] “IBR-DTN project.” https://trac.ibr.cs.tu-bs.de/project-cm-2012-ibrdtn.

[20] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” in *Mobile computing*, pp. 153–181, Springer, 1996.

[21] A. Tonnesen, “Olsrd: Implementation code of the olsr,” *Available on line at http://www. olsr. org*.

[22] M. H. Jain and R. Patra, “Implementing delay tolerant networking,” 2003.

[23] S. Burleigh, “Interplanetary overlay network: An implementation of the dtn bundle protocol,” in *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, pp. 222–226, IEEE, 2007.

[24] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, “Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation,” *Electronic Communications of the EASST*, vol. 37, 2011.

[25] W.-B. Pöttner, J. Morgenroth, S. Schildt, and L. Wolf, “Performance comparison of dtn bundle protocol implementations,” in *Proceedings of the 6th ACM Workshop on Challenged Networks*, CHANTS ’11, (New York, NY, USA), pp. 61–64, ACM, 2011.

[26] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, “Ibr-dtn: An efficient implementation for embedded systems,” in *Proceedings of the Third ACM Workshop on Challenged Networks*, CHANTS ’08, (New York, NY, USA), pp. 117–120, ACM, 2008.

[27] “DTN Ref. Implementation.” http://www.dtnrg.org/wiki/Code.

[28] J. Lorincz, N. Ukic, and D. Begusic, “Throughput comparison of aodv-uu and dsr-uu protocol implementations in multi-hop static environments,” in *2007 9th International Conference on Telecommunications*, pp. 195–202, June 2007.