Fast and On-demand Network-wide Telemetry via Cluster Based Path Planning

Du Chen[1], Deyun Gao[1], Wei Quan[1], Chuan Heng Foh[2], Hongke Zhang[1]*

[1] School of Electronic and Information Engineering, Beijing Jiaotong University, China
[2] 5GIC & 6GIC, Institute for Communication Systems (ICS), University of Surrey, U.K.
{duchen, gaody, weiquan, hkzhang}@bjtu.edu.cn, c.foh@surrey.ac.uk

Abstract

In-band Network Telemetry (INT) has profoundly promoted the network visibility. However, the existing solutions either face Maximum Transmission Unit (MTU) limitation or latency issue due to continuous insertion of INT metadata along long forwarding paths. In addition, conducting telemetry using user packets is prone to result in duplicate telemetry and fall short to achieve network-wide telemetry. In this work, we propose NetworkSight to comprehensively resolve these problems through cluster based path planning and source routing based on-demand forwarding. Specifically, a cluster based path planning algorithm is designed to generate several balanced short forwarding paths that cover the entire network. As a consequence, the MTU limitation is overcome and the telemetry latency can be reduced. Instead of operating on user packets, NetworkSight sends crafted probes built on source routing to travel synchronously over the generated forwarding paths. With these improvements, network-wide telemetry is achieved. Besides, we define a suite of telemetry primitives and develop corresponding APIs for users to flexibly express their telemetry requirements. Extensive experimental results show that NetworkSight can generate more balanced forwarding paths. The results show that it outperforms state-of-the-art mechanisms, where it reduces the telemetry latency by tens of milliseconds.

Keywords: Cluster, Network management, Network telemetry, Path planning, Source routing

1 Introduction

With the continuous expansion of network scale and the continuous appearance of emerging applications, network management has become more complicated [1]. The fusion of machine learning (ML) technologies into network management has profoundly promoted network intelligence and automation [2-3]. To make optimal control decisions via ML, the global view of the underlying network and more detailed device-internal states are essential. However, the conventional network monitoring mechanisms, such as SNMP [4] and NetFlow [5], either operate at coarse timescale or collect coarse-grained network states, which cannot provide real-time and fine-grained network monitoring. Other OpenFlow-based network monitoring mechanisms are restricted since OpenFlow is protocol-dependent and can only continuously increase the matching fields to support new protocols [6].

Recent advances in programmable network devices and Programming Protocol-Independent Packet Processors (P4) have reshaped the landscape of software-defined networking (SDN) by enabling the users to specify how devices to process packets [7]. As a potential application of P4, In-band Network Telemetry (INT) allows packets to query the switch-internal states along the forwarding path traversed [8]. Thanks to this design, INT enables a more fine-grained way for network monitoring and facilitates a lot of network applications, including load balancing [9-10], congestion control [11-12], anomaly detection [13-14], and others.

However, using user packets to piggyback the INT metadata (i.e., the network states collected using INT) will result in rapid growth in the packet size. As a result, the packet size may exceed Maximum Transmission Unit (MTU), which further incurs fragmentation or packet loss [15]. Besides, inserting INT metadata into user packets as header fields may increase the processing latency, which will affect the quality-of-service (QoS) of applications, especially for delay-sensitive applications. Several studies propose sampling based approaches to reduce the insertion of INT metadata into packets [16-18]. However, selective insertion of INT metadata on partial packets may miss important network events. Furthermore, long forwarding paths will also introduce high telemetry latency due to long transmission delay. Hence, we argue that the length of forwarding paths should be limited to avoid the packet size exceeding MTU and reduce telemetry latency.

Besides, the basic INT specification only defines telemetry for specific paths or flows using user packets, which is unable to achieve network-wide telemetry [19]. Specifically, different users may choose the same devices to transmit their data or some devices may never handle any user packets, which leads to duplication of telemetry or coverage incompleteness of the entire network. Recently, many works attempt to extend the framework of original INT to support network-wide telemetry [20-22]. In these works, crafted probes are sent to travel specific paths and collect the network states. In particular, heuristic algorithms, such as Euler trail-based path planning algorithm, are designed to generate several forwarding paths that cover the entire network [21]. However, these algorithms are committed to achieving full coverage of the network with the fewest

*Corresponding Author: Deyun Gao; E-mail: gaody@bjtu.edu.cn
DOI: 10.53106/160792642023112406016

number of forwarding paths, which often generate extremely long and unbalanced forwarding paths. Obviously, it is impossible for the probes to complete telemetry of these the unbalanced forwarding paths simultaneously in a short time, only leading to a decrease in telemetry performance. Hence, we see the benefit that several short balanced forwarding paths should be generated to achieve network-wide synchronous multipath telemetry.

In this paper, we extend our previous works in [23] and propose NetworkSight, a cluster based fast and on-demand network-wide telemetry framework, to comprehensively address these problems. We observe that a probe can return from one cluster head to the same or reach another cluster head within 4 hops. With this observation, we divide the network into several clusters based on the concept of minimum dominating set (MDS). Then, we start from a cluster member, check its neighbours, and generate corresponding forwarding paths based on different cases. Once all the cluster members and cluster heads are visited, several short balanced forwarding paths are generated. In addition, we leverage source routing to enable the crafted probes to traverse the corresponding forwarding paths. In summary, this paper makes the following contributions:

- We propose a novel telemetry framework named NetworkSight, which sends source routing enabled crafted probes to traverse the specific forwarding paths, supporting fast and on-demand telemetry.
- We design a cluster based path planning algorithm to generate several short balanced forwarding paths, which allows querying of any node or link within 4 hops, leading to significant reduction of telemetry latency.
- We show the feasibility of our solution by presenting a suite of telemetry primitives and developing corresponding APIs to allow the users to flexibly express various telemetry requirements.
- We implement a prototype of NetworkSight on BMv2 and study the performance of NetworkSight through extensive experiments to show the performance advantages of NetworkSight over state-of-the-art mechanisms.

The remainder of this paper is organized as follows. In Section 2, we summary the related work of INT. The design principles of NetworkSight are presented in Section 3 along with the challenges to design a fast and on-demand network-wide telemetry system and how NetworkSight overcomes these challenges. In Section 4, we evaluate the proposed mechanism with extensive experiments and discuss the experimental results. Finally, we draw conclusions and highlight achievements in Section 5.

2 Related Work

In 2015, Kim *et al.* proposed the seminal work of INT, which allows packets to collect switch-internal states as they traverse the network [8]. Over the following years, many efforts have been devoted to improving the original INT framework to achieve higher telemetry accuracy with lower overhead. In this paper, we roughly summary them into two categories: Passive INT and Active INT.

Passive INT. Similar to the original INT framework, the Passive INT mechanisms also utilize user packets to collect network states, which incurs extra network overheads. Sampling based solutions are commonly used to tackle the issue of relatively high network overhead resulting from per-packet operation. Specifically, sINT [16] and Sel-INT [17] respectively design sampling rate determination scheme to adjust the INT header and/or metadata insertion rate to reduce the network overhead. Apart from rate based sampling, FS-INT further proposes an event based sampling mechanism [18]. However, selective insertion of INT metadata on partial packets may miss important network events.

Beside sampling based approaches, one other approach focuses on multilayer telemetry designs [24-26]. Specifically, Anand *et al.* propose POINT to observe IP and optical layers simultaneously [24]. Similarly, Niu *et al.* propose a multilayer telemetry system ML-INT, which samples partial packets from an IP flow to conduct INT-related operations [25]. Pan *et al.* further take into consideration of privacy and security issues and propose a privacy-preserving multilayer monitoring system based on INT [26].

In general, most of the Passive INT mechanisms mainly rely on TCP/UDP encapsulation to work, which brings about a dilemma between the space for INT insertion and MTU limitation. Moreover, different users may choose the same devices to transmit their data or some devices may never handle any user packets, which leads to duplication of telemetry or coverage incompleteness of the entire network. It is thus necessary to continue the investigation of the orchestration problem of INT and make an attempt to improve the operation [27].

Active INT. Different from Passive INT mechanisms, Active INT mechanisms actively send crafted probes to collect network states. For example, to achieve network-wide telemetry, HULA uses the ToR switches to flood probes to the uplinks [20]. However, HULA introduces high bandwidth consumption because multiple probes may traverse the same link(s).

Another approach focuses on designing heuristic algorithms to generate several forwarding paths that cover the entire network. For example, Pan *et. al* propose INT-path to utilize Euler-trail to generate forwarding paths for each pair of nodes with odd degree [21]. Although INT-path generates several non-overlapped forwarding paths, it fails to take path length into consideration. In other words, INT-path may generate several unbalanced forwarding paths, which results in the loss of synchronization of multipath telemetry. Furthermore, the network-wide telemetry latency is closely associated with the arriving time of the last probe. Long forwarding paths will only result in the late arrival of the probes, decreasing both timeliness and accuracy of the telemetry results.

NetView uses switch-level telemetry to estimate link-level states and converts the goal of covering the entire network into covering all the nodes [22]. Different algorithms are designed to generate as few probes as possible. However, only one server is used to inject probes into and collect probes from the network, which will introduce extra delays because the probes need to return back to the server after completing telemetry.

In summary, using crafted probes offers two-fold benefits. Firstly, Active INT mechanisms provide the possibility of overcoming MTU limitation through sending small size probes, which remain substantial space for INT insertion. Secondly, guiding probes to travel along the pre-planned forwarding paths by source routing can easily achieve network-wide telemetry. Despite these research efforts towards network-wide path planning, little has been done to generate balanced paths to achieve synchronous multipath telemetry.

3 NetworkSight Design

In this section, we first discuss the requirements to design a fast and on-demand network-wide telemetry system. Then, the overview of and main components of NetworkSight are presented. Finally, we elaborate how these challenges are comprehensively resolved via cluster based path planning and source routing based on-demand forwarding.

3.1 Telemetry Requirements

The key to designing a fast and on-demand network-wide telemetry system is optimal path planning and probe coordinating. The main design requirements are summarized as follows.

Full coverage. The system should cover all the links to provide a complete network visibility. In other words, path planning is needed to generate several forwarding paths that cover the entire network. In order to generate such forwarding paths, the same link may be multiplexed when conducting path planning. Since the multiplexed links will introduce duplicate telemetry, path overlapping should be avoided to minimize the redundancy.

Fast convergence. The convergence time refers to the time range between when a probe is sent out and when the corresponding INT report arrives at the collector, which mainly depends on two aspects. Firstly, long forwarding paths in general introduce long transmission delay, which incurs non-negligible delay in the arrival of probes. Thus, the path length should be limited when conducting path planning. Secondly, INT-related operations on the devices will introduce extra delay. For this reason, the corresponding match-action tables should be concise and efficient to process.

Low overhead. To perform telemetry, probes are injected into data plane, which consumes link bandwidth. A straightforward way to reducing the bandwidth consumption of probes is to inject as few probes as possible into the network and make each probe carry more INT metadata. However, one probe carrying more metadata means it has to travel through a long forwarding path, which contradicts fast convergence and may incur MTU issue. Hence, a tradeoff exists between the number of probes and the length of forwarding paths.

Flexible telemetry. It is often that different users demand different telemetry requirements, such as telemetry frequency, telemetry object and telemetry item. In order to satisfy the differentiated telemetry requirements, the system should provide either comprehensive tuning capability or open APIs to enable flexible control of the underlying telemetry operations.

3.2 The Design Overview of NetworkSight

To meet the requirements, our proposed NetworkSight builds on two steps: cluster based path planning and source routing based on-demand forwarding. Figure 1 depicts the architecture of NetworkSight, which contains five components: Open APIs, Telemetry Synthesizer, Path Generator, Probe Manager and Data Analyzer.

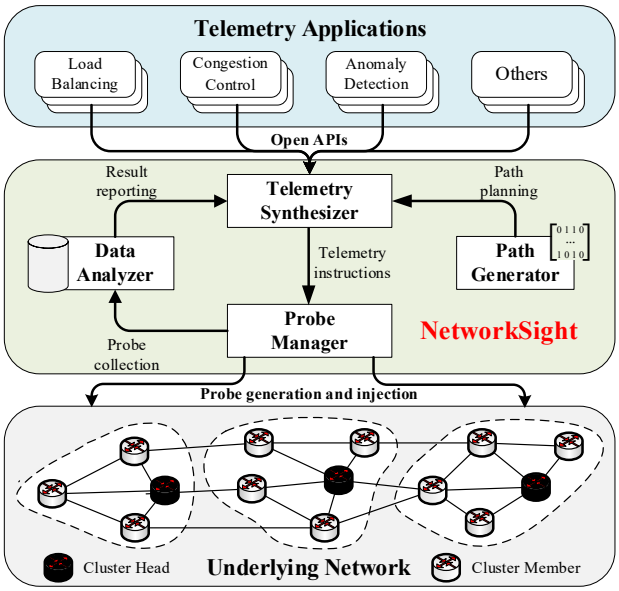

Figure 1. The architecture of NetworkSight

Open APIs. NetworkSight provides Open APIs for users to flexibly express their telemetry requirements. To accomplish this, we define a suite of telemetry primitives: (1) *Frequency()* donates the telemetry frequency. (2) *Node(Node: PortNumber)* and *Link(Source: Destination)* are used to indicate which nodes or links to telemetry, respectively. In particular, *Link(ALL: ALL)* refers to network-wide telemetry. (3) *Item()* specifies which states to collect. With these primitives, NetworkSight enables collecting the desired states of nodes or links at a specified frequency.

Telemetry Synthesizer. Once receiving the telemetry requirements from the Open APIs, Telemetry Synthesizer first queries the database whether there is historical data that satisfies the demands. If not, Telemetry Synthesizer merges the same telemetry requirements and generates the optimal telemetry instructions based on the path planning results provided by Path Generator. These instructions are then issued to Probe Manager for probe generation. In this way, fewer probes are generated and the bandwidth overhead is reduced.

Path Generator. For the purpose of full coverage and fast convergence, Path Generator adapts a cluster based path planning algorithm to generate several short balanced forwarding paths. Specifically, we divide a network into several clusters based on MDS through treating the dominating nodes and the dominated nodes as cluster heads and cluster members, respectively. Then, we go through all

the cluster members, check the neighbours of each cluster member, and generate corresponding forwarding paths based on different cases. The path planning results are provided to Telemetry Synthesizer to generate optimal telemetry policies.

Probe Manager. In NetworkSight, Probe Manager is responsible for generating probes according to the received telemetry instructions, injecting probes into and collecting probes from the underlying network. The probe can be roughly divided into two parts: packet header and INT header. Different from the user packets, the packet header of the probe only contains the Ethernet field, which further reduces the bandwidth consumption.

Data Analyzer. After the probes complete the telemetry tasks, Probe Manager collects them from the underlying network and submits them to Data Analyzer, which extracts important information from the raw data and generates corresponding telemetry reports. These reports are finally sent to the users through Telemetry Synthesizer, which are also used to update the database simultaneously.

In the following subsections, we will comprehensively introduce the cluster based path algorithm used by Path Generator along with how source routing enables crafted probes to travel the forwarding paths on-demand.

3.3 Cluster Based Path Planning

To keep every node and every link under monitoring, path planning is essential. As discussed in the previous subsections, the path length should be considered to avoid long forwarding paths when conducting path planning. We observe that a probe can return from one cluster head to the same or reach another cluster head within 4 hops. With this observation, we propose a cluster based path planning algorithm as Algorithm 1.

To classify the nodes in the network into cluster heads and cluster members, we first find a minimum dominating set for the network. For a graph (i.e., a network topology) G, we define the node set as N while the link set as L. If there is node set D and any node in $N - D$ has at least one neighbour in D, we call D a dominating set of G. Besides, a minimum dominating set of G refers to the dominating set which contains the least number of nodes. One thing should be noted that, one graph may have multiple minimum dominating sets. Here, we find a minimum dominating set for a graph through a greedy method. Specifically, we initial the cluster head set *CH* as an empty set. Besides, the unvisited node set *UN* and the candidate node set *CN* are both initialed as *N*. For each node in *CN*, we select the node with the maximum number of neighbours in *UN* (Line 1-6) and append it to *CH* (Line 14-15). Then, this node is subtracted from *UN* and *CN* while its neighbours are subtracted from *UN* (Line 16-18). We repeat these steps until *UN* is empty (Line 13-19). In this way, a minimum dominating set (i.e., the cluster head set) for the graph is found. The cluster division can be done through treating the dominating nodes and the dominated nodes as cluster heads and cluster members, respectively.

Algorithm 1. MDS based clustering algorithm

```
Input: A graph G(N, L)
Output: The clustering results Cluster
Initialize: CH ← ∅, UN ← N, CN ← N
function Searching_ Head (G, UN, CN)
    for node in CN do
        node.neighbour← neighbours in UN
    end for
    Return node with maximum node.neighbour
end function
function Assigning_ Member (G, UN, CH)
    for node in CH do
        node.neighbour← neighbours in UN
    end for
    Return node with minimum node.neighbour
end function
while UN != ∅ do
    head← Searching _Head (G, UN, CN)
    CH← CH + head
    CN← CN – head
    head.neighbour← neighbours in UN
    UN← UN – head – head.neighbour
end while
UN← N – CH
count← 0
while count < CH.length do
    head← Assigning_Member (G, UN, CH)
    head.neighbour← neighbours in UN
    Cluster[count]← {head, head.neighbour}
    CH← CH – head
    UN← UN – head.neighbour
    count← count + 1
end while
Return the clustering results Cluster
```

Since one dominated node may connect with more than one dominating node, how to assign cluster members for a cluster head remains unsolved. To make the clusters more balanced, we adopt the opposite approach to that of searching a minimum dominating set. Here, we start from the cluster head with minimum number of neighbours in *UN* (Line 7-12) and assign cluster members for it (Line 23-25). Similarly, we subtract this cluster head from *CH* and the corresponding cluster members from *UN* (Line 26-27). Then, we continue to assign cluster members for the cluster head in the same way until *CH* is empty (Line 22-29). Finally, the clustering results are obtained.

We now conduct path planning based on the clustering results. Since each cluster member is directly connected to its cluster head, generating forwarding paths for each pair of cluster member and the corresponding cluster head can cover all the nodes. However, links may exist between two cluster members. For example, link $L2$ between $M1$ and $M2$ in the same cluster (shown in Figure 2(a)) and link $L6$ between $M3$ and $M4$ in different clusters (shown in Figure 2(b)). In addition, one cluster member or one cluster head may connect with another cluster head, such as link $L5$ and link $L8$ depicted in Figure 2(b). In order to achieve network-wide telemetry, these links should also be covered. Here, we conduct path planning for these links into different cases.

For intra-cluster case, we aggregate *L*2 with the links *L*1 and *L*3, which connect *M*1 and *M*2 to their cluster head *H*1, respectively. Hence, only one forwarding path *H*1-*M*1-*M*2-*H*1 is generated for the links *L*1, *L*2, and *L*3. For inter-cluster case, NetworkSight starts from a cluster member, such as cluster member *M*3 of cluster head *H*2, and searches whether there is another cluster head connected to it. If the answer is "*Yes*", such as cluster head *H*3, NetworkSight further checks whether there is a link between *H*2 and *H*3. If there is, such as link *L*8, NetworkSight generates a forwarding path *H*2-*M*3-*H*3-*H*2 for the links *L*4, *L*5, and *L*8. Otherwise, NetworkSight generates a forwarding path *H*2-*M*3-*H*3 for the links *L*4 and *L*5. If there are no other cluster heads connected to *M*3, NetworkSight then checks whether there is a cluster member of another cluster connected to it. If there is, such as *M*4, NetworkSight generates a forwarding path *H*2-*M*3-*M*4-*H*3 for *L*4, *L*6, and *L*7 (*L*7 is the link between *M*4 and its cluster head *H*3). If there are neither other cluster heads nor cluster members connected to *M*3, NetworkSight directly generates a forwarding path for link *L*4. Finally, we generate forwarding paths for the links between cluster heads that are not visited. In this way, all the links are considered and the generated forwarding paths are always within 4 hops.

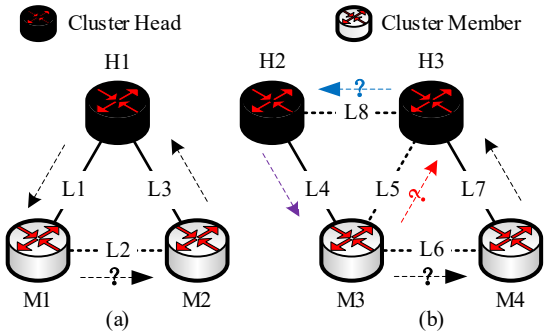

Figure 2. An example of path aggregating

Based on the above example, we further present the complete path planning process in Algorithm 2. We start from a cluster head, such as *a*, and search from each cluster member in this cluster, such as *b*. If *b* has no neighbours expect *a*, we generate a forwarding path for the link between *a* and *b* (Line 3-5). If *b* has other neighbours, we further search from each neighbour of *b*, such as *c*. If *c* is another cluster head, we further check whether *c* is a neighbour of *a*. If *c* is and the link *l_ca* is not used to formulate another forwarding path, we generate a forwarding path *p_abca* for the links *l_ab*, *l_bc* and *l_ca* (Line 8-11). Otherwise, if *c* is not a neighbour of *a* or the link *l_ca* is used, we only generate a forwarding path *p_abc* for the links *l_ab* and *l_bc* (Line 12-15). If *c* is another cluster member of *a*, we generate a forwarding path *p_abca* for the links *l_ab*, *l_bc* and *l_ca* (Line 16-20). If *c* is neither another cluster head nor another cluster member of *a*, *c* must be a cluster member in another cluster. We donate the cluster head of *c* as *d* and generate a forwarding path *p_abcd* for the links *l_ab*, *l_bc* and *l_cd* (Line 21-26). Once finishing the searching for every cluster head and deleting the visited links from the link set *L* of graph *G*, the remaining links are the links between two cluster heads. We just generate corresponding forwarding paths for these links (Line 32-34). It is worth mentioning that we always start searching from the cluster heads or cluster members with smaller sequence number to avoid generating duplicate forwarding paths for the same links (Line 17 and line 23).

Algorithm 2. Cluster based path planning algorithm
Input: A graph *G*(*N*, *L*), the clustering results *Cluster*
Output: The path planning results *Path*
Initialize: *Path* ← ∅

```
1:  for a in CH do
2:      for b in a.member do
3:          if b.neighbour – a == ∅ then
4:              Path = Path + p_aba
5:              L = L – l_ab
6:          else
7:              for c in b.neighbour – a do
8:                  if c is in CH then
9:                      if l_ca is in L then
10:                         Path = Path + p_abca
11:                         L = L – l_ab - l_bc - l_ca
12:                     else
13:                         Path = Path + p_abc
14:                         L = L – l_ab - l_bc
15:                     end if
16:                 else if c is in a.member then
17:                     if c > b then
18:                         Path = Path + p_abca
19:                         L = L – l_ab - l_bc - l_ca
20:                     end if
21:                 else
22:                     d ← the cluster head of c
23:                     if d > a then
24:                         Path = Path + p_abcd
25:                         L = L – l_ab - l_bc - l_cd
26:                     end if
27:                 end if
28:             end for
29:         end if
30:     end for
31: end for
32: for l in L do
33:     Path = Path + l
34: end for
35: Return the path planning results Cluster
```

With the proposed cluster based path planning algorithm, several short balanced forwarding paths are generated. In the following, we will introduce how to enable probes to traverse these forwarding paths.

3.4 Source Routing based On-demand Forwarding

Generally, the intermediate nodes take the responsibilities for routing and addressing while the source node only provides a destination address. On the contrary, source routing allows the source node to specify the complete routing information for packets [28]. Benefit from this ability, NetworkSight utilizes source routing to forward probes on-demand with the routing information being embedded

into the probes. The format of the NetworkSight probes is depicted as Figure 3.

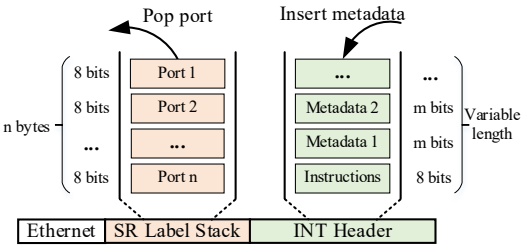

Figure 3. The format of the NetworkSight probes

To indicate the coming packet is a NetworkSight probe, the EtherType field in Ethernet header is set to 0x1234. Above the Ethernet header, NetworkSight encapsulates the source routing label stack, which contains the complete routing information. Specifically, each source routing label occupies 8 bits, indicating through which port to forward the probe. Next, NetworkSight allocates a variable-length INT header, which consists of 8-bit telemetry instructions and variable-number INT metadata. Here, the telemetry instructions are used to indicate which states (i.e., the telemetry items) to collect. Since the length of each forwarding path is within 4 hops, the source routing label stack takes up no more than 32 bits. Besides, the INT header also occupies limited space in a probe because the types of telemetry items are limited. In this way, the probe size is full under control, which never exceeds MTU.

Once receiving a packet, Networksight parses it as follows. Firstly, the packet headers are processed to check whether the packet is a probe. If the value of EtherType field equals to 0x1234, it is a probe and INT-related operations are conducted. Otherwise, it is a user packet and NetworkSight forwards it directly. For a probe, corresponding INT metadata is inserted into it according to the telemetry instructions. Then, a source routing label is popped and the probe is sent to next hop. Finally, the probe is sent to the collector at the sink node.Supported by source routing, NetworkSight enables crafted probes to traverse the generated forwarding paths that cover the entire network. Besides, the probe always travels a short forwarding path, significantly reducing network-wide telemetry latency. In this way, fast network-wide telemetry is achieved.

4 Evaluation

In this section, we evaluate the performance of NetworkSight. We first evaluate the cluster based path planning algorithm on a server with 64GB RAM and Intel(R) Xeon(R) Silver 4214R 2.40GHz CPU, running Ubuntu 18.04. Then, Mininet is used to establish different network topologies to compare the telemetry performance of NetworkSight with that of state-of-the-art mechanisms in terms of telemetry latency, telemetry overhead. Besides, we build a testbed to validate that NetworkSight can be deployed in practice.

4.1 Evaluation of Path Planning Algorithm

Execution time. We run the programs for 100 times and calculate the average execution time of path planning for different network topologies, respectively. Each network topology is provided with its adjacency matrix. As shown in Figure 4, the execution time of path planning increases with the expansion of the network size in both NetworkSight and NetView while it does not exhibit this trend in INT-path. This is because the execution time of path planning in NetworkSight and NetView is mainly associated with the speed of accessing to the adjacency matrix. However, in INT-path, the execution time of path planning depends on not only the speed of accessing to the adjacency matrix but also the number of odd nodes. Besides, NetworkSight can complete path planning faster than both NetView and INT-path.

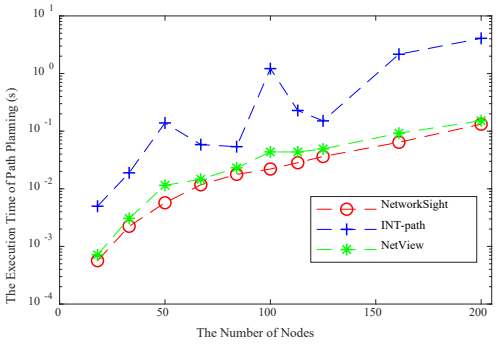

Figure 4. Impact of network size on the execution time of path planning

Path length. Here, we present the longest path length among all forwarding paths generated for a given network topology using different path planning algorithms. As depicted in Figure 5, the forwarding paths generated by NetworkSight are always within 4 hops regardless of the network size. Unlike NetworkSight, the longest path length among all forwarding paths generated by INT-path and NetView varies with the network size. In particular, the longest forwarding path generated by INT-path even reaches as high as 84 hops in a 50-node network. The reason is that INT-path relies heavily on odd nodes to conduct path planning. When there are fewer odd nodes in the network, INT-path easily generates an enormously long forwarding path. Furthermore, INT-path will generate only one forwarding path that cover all the nodes when there are none or two odd nodes in the network. On the other hand, NetView uses only one server to inject probes into and collect probes from the network. Even if shortest path between the server and a node to telemetry is used, NetView will also generate long forwarding paths when the network diameter is large. Different from INT-path and NetView, NetworkSight proactively controls the length of forwarding path when conducting path planning. Consequently, NetworkSight generates much shorter forwarding paths.

Beside the maximum value, the variance of path length is calculated to make further comparison. Figure 6 shows that NetworkSight generates more balanced forwarding paths with much smaller variance of path length. On the contrary, the other two mechanisms, INT-path and NetView, fail to

eliminate differences in path length. As a result, neither INT-path nor NetView can achieve synchronous multipath telemetry.

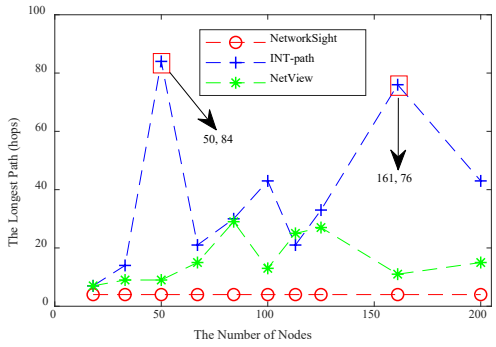

Figure 5. Impact of network size on the path length

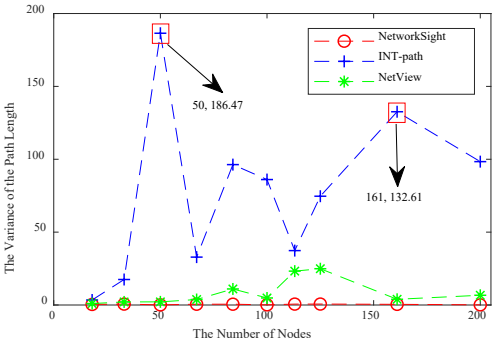

Figure 6. The variance of path length

Path number. Figure 7 illustrates the number of the forwarding paths generated by different path planning algorithms. NetworkSight generates much more forwarding paths than INT-path and NetView. This is because NetworkSight proactively controls the path length and only generates shorter paths (no more than 4 hops) when conducting path planning. On the contrary, INT-path and NetView prefer to generate fewer forwarding paths and fall short to take the path length into account. Besides, the number of the forwarding paths generated by NetView is often more than that of INT-path. The reason is that NetView generates a shortest forwarding path for a fixed source and a random destination. Hence, the number of forwarding paths generated by NetView depends heavily on the number of nodes in the network.

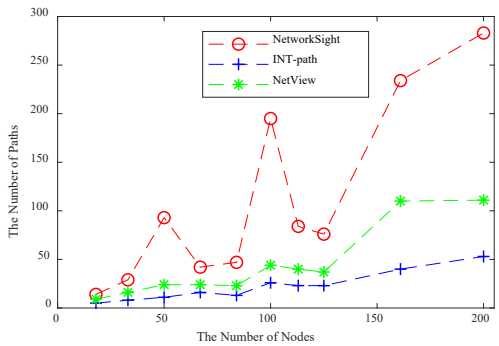

Figure 7. Impact of network size on the path number

4.2 Evaluation of Telemetry Performance

To further evaluate the telemetry performance of NetworkSight, we carry out extensive simulations to compare NetworkSight with state-of-the-art mechanisms. Here, we use Mininet to establish different network topologies with the number of nodes ranging from 18 to 200 [29]. Besides, the telemetry latency and telemetry overhead are measured in each network topology. In addition, BMv2 switches are used to enable NetworkSight to define customized packet headers and 22-byte INT metadata per-hop is inserted into the probes.

Telemetry latency. For each network topology, we measure the end-to-end telemetry latency of all probes using NetworkSight, INT-path and NetView, respectively. Figure 8 shows that the end-to-end telemetry latency of each probe varies even if using the same telemetry mechanism to inject the probes into the same network. This is because the probes travel different forwarding paths in the network. Clearly, the median value of the box of NetworkSight is always smaller than that of both INT-path and NetView for the same network topology. This reveals that most of the probes injected by NetworkSight can complete telemetry in a much shorter time. Similarly, the maximum value of the box of NetworkSight is also the smallest, which demonstrates that NetworkSight achieves a lowest network-wide telemetry latency. For different network topologies, this conclusion still holds. As mentioned earlier, the telemetry latency is tightly associated with path length and long forwarding paths only result in a high telemetry latency. The short balanced forwarding paths generated by NetworkSight can significantly reduce the end-to-end telemetry latency for all probes and thus achieve fast network-wide telemetry.

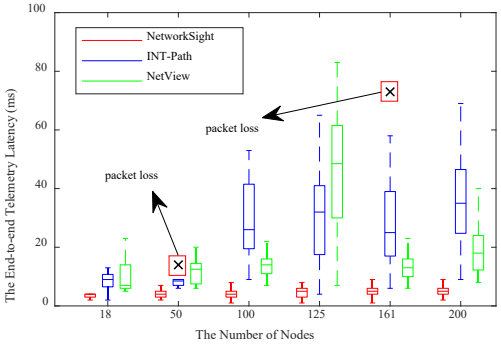

Figure 8. The telemetry latency of different mechanisms

On the contrary, both INT-path and NetView do not proactively limit the path length and thus generate more unbalanced forwarding paths when conducting path planning. As a result, the end-to-end telemetry latency of each probe fluctuates more dramatically in INT-path and NetView. For example, in a 100-node network, the probe needs 9 ms to complete telemetry for a short forwarding path while 53 ms is cost for a long forwarding path using INT-path. For NetView, the end-to-end telemetry latency ranges from 7 ms to 22 ms. Besides, packet loss occurs in the cases of 50-node and 161-node networks using INT-path. The reason of the packet loss is that INT-path generates enormously long forwarding paths for these networks, which results in the probe size exceeding MTU due to conducting telemetry on these forwarding paths.

Consequently, these forwarding paths hinder the probes from completing telemetry and reaching the collector. Eventually, INT-path fails to achieve network-wide telemetry.

Telemetry overhead. To evaluate the telemetry overhead of NetworkSight, we measure the bandwidth consumption of NetworkSight and make comparisons with that of INT-path and NetView. Figure 9 shows that the bandwidth consumption of each telemetry mechanism increases with the expansion of network size.

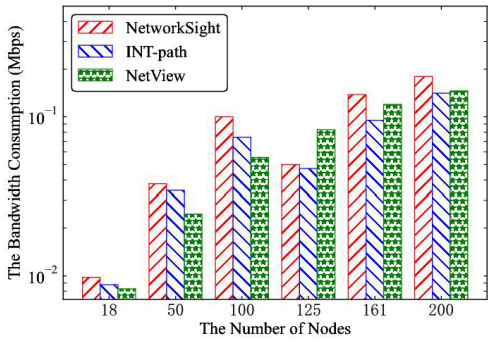

Figure 9. The bandwidth consumption of different mechanisms

In general, the bandwidth consumption is proportional to the number of the probes injected into the network. Although NetworkSight injects more probes into the network than INT-path and NetView do, the size of each probe in NetworkSight is much smaller than that of the other two telemetry mechanisms. Specifically, the probes in each telemetry mechanism contain Ethernet header, source routing label stack and INT header. The source routing label stack and INT header of the probes generated by NetworkSight occupy less space, which are only h bytes and $22\times h$ bytes, respectively. Here, h refers to the hops a probe traversed, which is no more than 4. On the other hand, INT-path and NetView reserve more space for source routing label stack (64 bytes in INT-path while 2 bytes for each hop in NetView). Besides, an 8-byte UDP header and a 20-byte IP header are encapsulated in the probes simultaneously, which further increase the bandwidth consumption. As shown in Figure 6, the variance of path length in INT-path and NetView are relatively bigger, making the probe size hard to estimate. As a result, the bandwidth consumption of NetworkSight is not much higher than that of INT-path and NetView for all network topologies.

In summary, NetworkSight can generate more balanced forwarding paths and thus achieves fast network-wide telemetry. Besides, the bandwidth consumption of NetworkSight is relatively low, which makes it practical for deployment.

4.3 Testbed Testing

To study the feasibility of NetworkSight, we build a 3-pod Fat-Tree network, which contains 15 switches and 6 hosts. Besides, every two switches are connected by a 1G/s link while each host is connected with the corresponding ToR by a 500Mb/s link.

Varying telemetry frequency. To evaluate the performance of NetworkSight under different telemetry frequency, fixed telemetry items (1, 5, 10) are collected for every switch and the telemetry overhead is measured. Besides, each host periodically communicates with a host in another pod at the speed of 100 Mbps, generating the background traffic. As shown in Figure 10, the bandwidth consumption grows with the increase of the telemetry frequency regardless the types of telemetry items collected. Besides, the bandwidth consumption can be neglected when the telemetry frequency is small. However, the bandwidth consumed at the telemetry frequency of 1000 Hz is only slightly greater than that at the telemetry frequency of 100 Hz. This is because NetworkSight cannot complete the network-wide telemetry task in such a short time (i.e, 1 ms).

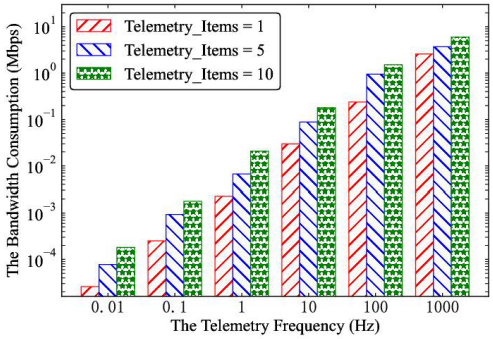

Figure 10. The bandwidth consumption under different telemetry frequency

Varying background traffic. Now we focus on the performance of NetworkSight under different background traffic. Similarly, fixed telemetry items (1, 5, 10) are collected for every switch and each host periodically communicates with a host in another pod. We make the communication speed vary from 0 to 500 Mbps to generate different background traffic and measure the network-wide telemetry latency. The telemetry frequency is fixed as 10 Hz and the average network-wide telemetry latency is calculated. Figure 11 shows that the network-wide telemetry latency increases rapidly when the background traffic is lower than 500 Mbps. However, when the background traffic is 500 Mbps, packet loss occurs and NetworkSight falls short to complete network-wide telemetry due to congestion.

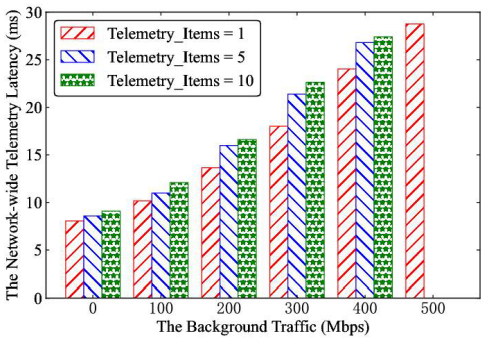

Figure 11. The network-wide telemetry latency under different background traffic

The above experiments on the testbed confirm that NetworkSight can provide fast and on-demand telemetry

with a low bandwidth consumption at a high telemetry frequency when the background traffic is small. However, it will cost more time for NetworkSight to complete network-wide telemetry when network congestion occurs. For these reasons, we suggest using NetworkSight at the telemetry frequency no exceeding 100 Hz in the scenario where the network traffic load is relatively low.

5 Conclusion

In this paper, the problem of fast and on-demand network-wide monitoring was addressed. A novel telemetry framework named NeworkSight was proposed. Owing to the short balanced forwarding paths generated by the cluster based path planning and the crafted probes built on source routing, we showed that NetworkSight achieved fast and on-demand synchronous multipath telemetry in a more efficient way. Besides, the telemetry primitives and corresponding APIs enabled various applications to express the telemetry requirements more flexibly. Extensive experiments revealed that NetworkSight outperformed state-of-the-art mechanisms, generating more balanced forwarding paths and significantly reducing network-wide telemetry latency with acceptable telemetry overhead.

Acknowledgements

This work is supported in part by the Fundamental Research Funds for the Central Universities (Grant No. 2022YJS123), the National Key R&D Program of China (Grant No. 2022YFB2901900), the National Natural Science Foundation of China (Grant No. 61971028, 62322102 and 62372035), the Joint Fund of Ministry of Education (Grant No. 8091B032222), and in part by the Beijing Nova Program (Grant No. 20220484084). The authors would like to acknowledge the support of the 5GIC & 6GIC members for this work.

References

[1] C.-X. Wang, X. You, X. Gao, X. Zhu, Z. Li, C. Zhang, H. Wang, Y. Huang, Y. Chen, H. Haas, Y. Chen, H. Haas, J. S. Thompson, E. G. Larsson, M. D. Renzo, W. Tong, P. Zhu, X. Shen, H. V. Poor, L. Hanzo, On the Road to 6G: Visions, Requirements, Key Technologies, and Testbeds, *IEEE Communications Surveys & Tutorials*, Vol. 25, No. 2, pp. 905-974, Second Quarter, 2023.

[2] S. Chen, H. Wen, J. Wu, Artificial Intelligence Based Traffic Control for Edge Computing Assisted Vehicle Networks, *Journal of Internet Technology*, Vol. 23, No. 5, pp. 989-996, September, 2022.

[3] Y.-W. Chen, J.-Z. You, Effective Radio Resource Allocation for IoT Random Access by Using Reinforcement Learning, *Journal of Internet Technology*, Vol. 23, No. 5, pp. 1069-1075, September, 2022.

[4] J. D. Case, M. Fedor, M. L. Schoffstall, J. Davin, Simple Network Management Protocol (SNMP), RFC1157, May, 1990. https://datatracker.ietf.org/doc/rfc1157/

[5] B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC3954, October, 2004. https://datatracker.ietf.org/doc/rfc3954/

[6] W. L. da Costa Cordeiro, J. A. Marques, L. P. Gaspary, Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management, *Journal of Network and Systems Management*, Vol. 25, No. 4, pp. 784-818, October, 2017.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming Protocol-Independent Packet Processors, *ACM SIGCOMM Computer Communication Review*, Vol. 44, No. 3, pp. 87-95, July, 2014.

[8] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker, In-band Network Telemetry via Programmable Dataplanes, *Proc. ACM SIGCOMM*, London, UK, 2015, pp. 1-2.

[9] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford, Clove: Congestion-Aware Load Balancing at the Virtual Edge, *Proc. CoNEXT: Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, Incheon, Republic of Korea, 2017, pp. 323-335.

[10] W. Gao, J. Huang, N. Jiang, Z. Li, S. Zou, Z. He, J. Wang, HPLB: High Precision Load Balancing Based on In-band Network Telemetry in Data Center Networks, *Peer-to-Peer Networking and Applications volume*, Vol. 15, No. 6, pp. 2503-2515, November, 2022.

[11] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, M. Yu, HPCC: High Precision Congestion Control, *SIGCOMM'19: Proceedings of the ACM Special Interest Group on Data Communication*, Beijing, China, 2019, pp. 44-58.

[12] K. Liu, W. Quan, N. Cheng, W. Wu, Z. Xu, L. Guo, D. Gao, H. Zhang, Reliable PPO-Based Concurrent Multipath Transfer for Time-Sensitive Applications, *IEEE Transactions on Vehicular Technology*, Vol. 72, No. 10, pp. 13575-13590, October, 2023.

[13] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, Y. Liu, Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry, *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, Budapest, Hungary, 2020, pp. 1-9.

[14] Y. Zhang, T. Pan, Y. Zheng, M. Gao, H. Wang, T. Huang, Y. Liu, Automating Rapid Network Anomaly Detection with In-Band Network Telemetry, *IEEE Networking Letters*, Vol. 4, No. 1, pp. 39-42, March, 2022.

[15] Q. Yuan, F. Li, T. Pan, Y. Lai, Y. Gu, X. Wang, INT-Segment: MTU-Adaptive Single-Path In-Band Network-Wide Telemetry, *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, Lexington, KY, USA, 2022, pp. 1-11.

[16] Y. Kim, D. Suh, S. Pack, Selective In-band Network Telemetry for Overhead Reduction, *2018 IEEE 7th*

International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 2018, pp. 1-3.
[17] S. Tang, D. Li, B. Niu, J. Peng, Z. Zhu, Sel-INT: A Runtime-Programmable Selective In-band Network Telemetry System, *IEEE Transactions on Network and Service Management*, Vol. 17, No. 2, pp. 708-721, June, 2020.
[18] D. Suh, S. Jang, S. Han, S. Pack, X. Wang, Flexible Sampling-based In-band Network Telemetry in Programmable Data Plane, *ICT Express*, Vol. 6, No. 1, pp. 62-65, March, 2020.
[19] The P4.org Applications Working Group, *In-band Network Telemetry (INT) Dataplane Specification*, June, 2020. https://p4.org/p4-spec/docs/INT v2 1.pdf
[20] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, Hula: Scalable Load Balancing Using Programmable Data Planes, *SOSR'16: Proceedings of the Symposium on SDN Research*, Santa Clara, CA, USA, 2016, pp. 1-12.
[21] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, Y. Liu, INT-path: Towards Optimal Path Planning for In-band Network-wide Telemetry, *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Paris, France, 2019, pp. 487-495.
[22] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, J. Wu, Netview: Towards On-demand Network-wide Telemetry in the Data Center, *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland, 2020, pp. 1-6.
[23] D. Chen, D. Gao, C. H. Foh, H. Yan, CFINT: Cluster Based Fast In-band Network-wide Telemetry in 6G-enabled Networks, *2022 IEEE Globecom Workshops (GC Wkshps)*, Rio de Janeiro, Brazil, 2022, pp. 1699-1704.
[24] M. Anand, R. Subrahmaniam, R. Valiveti, POINT: An Intent-Driven Framework for Integrated Packet-Optical In-Band Network Telemetry, *2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, 2018, pp. 1-6.
[25] B. Niu, J. Kong, S. Tang, Y. Li, Z. Zhu, Visualize Your IP-Over-Optical Network in Realtime: A P4-Based Flexible Multilayer In-Band Network Telemetry (ML-INT) System, *IEEE Access*, Vol. 7, pp. 82413-82423, June, 2019.
[26] X. Pan, S. Tang, S. Liu, J. Kong, X. Zhang, D. Hu, J. Qi, Z. Zhu, Privacy-Preserving Multilayer In-Band Network Telemetry and Data Analytics: For Safety, Please do Not Report Plaintext Data, *Journal of Lightwave Technology*, Vol. 38, No. 21, pp. 5855-5866, November, 2020.
[27] R. Hohemberger, A. G. Castro, F. G. Vogt, R. B. Mansilha, A. F. Lorenzon, F. D. Rossi, M. C. Luizelli, Orchestrating In-Band Data Plane Telemetry with Machine Learning, *IEEE Communications Letters*, Vol. 23, No. 12, pp. 2247-2251, December, 2019.
[28] C. A. Sunshine, Source Routing in Computer Networks, *ACM SIGCOMM Computer Communication Review*, Vol. 7, No. 1, pp. 29-33, January, 1977.
[29] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The Internet Topology Zoo, *IEEE Journal on Selected Areas in Communications*, Vol. 29, No. 9, pp. 1765-1775, October, 2011.

Biographies

Du Chen is a Ph.D. student with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China. His research interests include future Internet, multipath TCP, and programmable data plane.

Deyun Gao is a Full Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China. His research interests include Internet of Things, vehicular networks, software defined networking, and programmable data plane.

Wei Quan is a Full Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China. His research interests include key technologies for network analytics, future Internet, 6G networks, and vehicular networks.

Chuan Heng Foh is a Senior Lecturer with the University of Surrey, Guildford, U.K. His research interests include protocol design and performance analysis of various computer networks including wireless local area and mesh networks, mobile ad hoc and sensor networks, Internet of Things, 5G networks, and data center networks.

Hongke Zhang is a Full Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, where he directs the National Engineering Research Center of Advanced Network Technologies. His research has resulted in many papers, books, patents, systems, and equipment, in the areas of communications and computer networks.