Computers, Materials & Continua
DOI:10.32604/cmc.2022.022344
Article

Tech Science Press

Multi-Path Service Function Chaining for Mobile Surveillance of Animal Husbandry

Xi Chen[1,3], Tao Wu[2,*] and Mehtab Afzal[4]

[1]School of Computer Science and Engineering, Southwest Minzu University, Chengdu, 610041, China
[2]School of Computer Science, Chengdu University of Information Technology, Chengdu, 610225, China
[3]School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China
[4]Department of Computer Science and IT, The University of Lahore, Lahore, 54000, Pakistan
[*]Corresponding Author: Tao Wu. Email: wut@cuit.edu.cn
Received: 04 August 2021; Accepted: 07 September 2021

Abstract: Animal husbandry is the pillar industry in some ethnic areas of China. However, the communication/networking infrastructure in these areas is often underdeveloped, thus the difficulty in centralized management, and challenges for the effective monitoring. Considering the dynamics of the field monitoring environment, as well as the diversity and mobility of monitoring targets, traditional WSN (Wireless Sensor Networks) or IoT (Internet of Things) is difficult to meet the surveillance needs. Mobile surveillance that features the collaboration of various functions (camera, sensing, image recognition, etc.) deployed on mobile devices is desirable in a volatile wireless environment. This paper proposes the service function chaining for mobile surveillance of animal husbandry, which orchestrates multi-path multi-function (MPMF) chains to help mobile devices to collaborate in complex surveillance tasks, provide backup chains in case the primary service function chain fails due to mobility, signal strength, obstacle, etc., and make up for the defects of difficult deployment of monitoring facilities in ethnic areas. MPMF algorithm models both mobile devices and various functions deployed on them as abstract graph nodes, so that chains that are required to traverse various functions and hosting mobile devices can be orchestrated in a single graph-based query through modified and adapted Dijkstra-like algorithms, with their cost ordered automatically. Experiment results show that the proposed MPMF algorithm finds multiple least-costly chains that traverse demanded functions in a timely fashion on Raspberry Pi-equipped mobile devices.

Keywords: Service function chaining; orchestration; surveillance; shortest path

1 Introduction

Animal husbandry is the pillar industry in some ethnic areas in China. Affected by climate, grazing and other factors, animal husbandry areas often face the problems of grassland

degradation, destruction of plants, wandering away of herds, invasion of natural predators, rain and snow disasters, etc., which result in remarkable economic losses. In order to effectively reduce the losses caused by these problems, it is necessary to carry out continuous, long-term and dynamic monitoring over vast gazing lands and wide-spread livestock. However, the communication/networking infrastructure in ethnic areas is often underdeveloped, thus the difficulty in centralized management, and challenges for the effective monitoring.

At the same time, considering the complexity and dynamics of the field monitoring environment, as well as the diversity and mobility of monitoring targets (such as mobile herds, lost individuals, natural enemies, etc.), the requirements of mobility, collaboration and low power consumption are put forward for monitoring devices. Therefore, simply introducing fixed-position sensors/cameras to deploy traditional wireless sensor networks (WSN) [1,2] or Internet of things (IoT) [3,4] is difficult to meet the mobility needs [5–8], so it is necessary to implement the mobile surveillance over gazing lands and livestock in ethnic areas. On the one hand, it requires that the wireless communication mechanism used by mobile monitoring devices should have good robustness, and be able to adapt to bad communication conditions; On the other hand, considering the limited computing power and functions on the devices in the mobile wireless environment, mobile monitoring devices are also required to have the ability to cooperate with each other in more complex surveillance tasks. To briefly summarize, orchestration of functionally diverse monitoring devices plays a key role to fulfill mobile surveillance of animal husbandry, given underdeveloped monitoring and communication infrastructure in ethnic areas. The mobile wireless environment is quite volatile in that the established collaborative relationships (e.g., the hop-by-hop routing) vary over time. Even though chains of mobile devices can be made by orchestration, they might stop functioning or do not function as expected due to mobility, obstacles, signal strength, etc. Therefore, orchestration of mobile devices in wireless environment should consider backup chains in case the primary chain fails. Meanwhile, chains should also reflect the quality in addition to functions, and better ones should be automatically selected as preferred chains.

In order to face these challenges, this paper proposes service function chaining [9,10] for mobile surveillance of animal husbandry, which orchestrates multi-path multi-function chains to help mobile devices to cooperate in complex surveillance tasks.

The rest of this paper is organized as follows. Section 2 introduces the mobile device architecture and functional wireless network topology in an orchestration view. Section 3 gives the formal modeling of the mobile devices' orchestration. Section 4 specifies the algorithm that solves the MPMF problem. Experiments are conducted in Section 5. Related works are summarized in Section 6. And finally, this paper is concluded and future works are envisioned in Section 7.

2 Mobile Device Architecture and Functional Wireless Network Topology

2.1 Mobile Device Architecture

In order to accomplish the service function chaining of various functions, a mobile device is roughly divided into 3 layers as shown in Fig. 1, namely, from bottom up, HW (hardware) layer, OS (operating system) layer, and orchestration layer, as we can see from Fig. 1. HW layer is the mobile hardware platform. Drones, smart cars, etc., with Raspberry Pi can be chosen to serve this purpose. HW layer provides mobility and various functions, e.g., camera, sensors, etc. We choose smart cars as the hardware platform in this paper. OS layer provides the operating system and basic software capabilities such as network, compute, and storage resources. In this paper, Raspbian OS is chosen to accommodate software capabilities, since it is lightweight and

Linux-compatible. Orchestration layer provides two major functionalities, namely, the collector, which collects information such as network topologies, various metrics that reflect performances of mobile devices, etc., and the orchestrator, which accepts service function chaining request and optimally chains demanded functions as per information gathered by the collector. Based on the previous discussion, we give the formal definition of the mobile device in this paper.

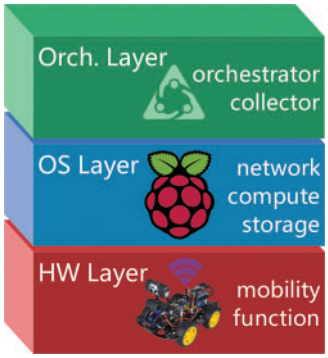

Figure 1: The mobile device architecture

Definition 1. The mobile device: The mobile device is defined as $v_i = (F_i, Q_i)$, where $F_i = \{f_{i,1}, f_{i,2}, \cdots, f_{i,||F_i||}\}$ represents functions deployed on the device ($f_{i,j}$ stands for the j-th function on the i-th device), such as picture shooting, sensing, etc., and $Q_i = \{q_{i,1}, q_{i,2}, \cdots, q_{i,||Q_i||}\}$ represents qualities of various metrics of the device ($q_{i,j}$ stands for the j-th metric's quality on the i-th device), such as bandwidth, delay, etc.

2.2 Functional View of the Wireless Network Topology

To orchestrate various functions deployed on mobile devices in a wireless network, one should investigate the functional view of the wireless network topology. Usually, one device is deployed with at least one function, such as photo-shooting, video surveillance, humidity sensing, etc., in addition to data-forwarding, as we can see from Fig. 2. To model functional view compatible with conventional topological view, functions can also be seen as nodes, which are directly connected with the hosting mobile device that are abstracted as nodes themselves. That is to say, functions are virtually one hop away from the hosting device, and are mutually connected through hosting devices, forming a cluster-like structure around the mobile device. Meanwhile, in a decentralized ad hoc wireless network, mutual links between mobile devices are rather temporary due to mobility, obstacles, signal strength, etc. It is usually a multi-hop network from the source to the destination device. Links are effective with devices staying within one another's radio coverage, but ineffective if outside, forming a mesh-like structure where one device could have multiple radio links with others, and communicate with others hop-by-hop. To summarize, the functional view of the mobile wireless network topology, when functions are treated as nodes like mobile devices, is a mesh-cluster structure, which is multi-hop globally for data streams between mobile devices, and single-hop locally for data streams between functions and the hosting device. For orchestration between functions, especially between those deployed on different mobile devices, data streams traverse in a function-device-function-device-function … interleaved manner, similar to service function chaining in NFV-enabled environments. Therefore, the orchestration of various functions can be modeled as service function chaining problem.

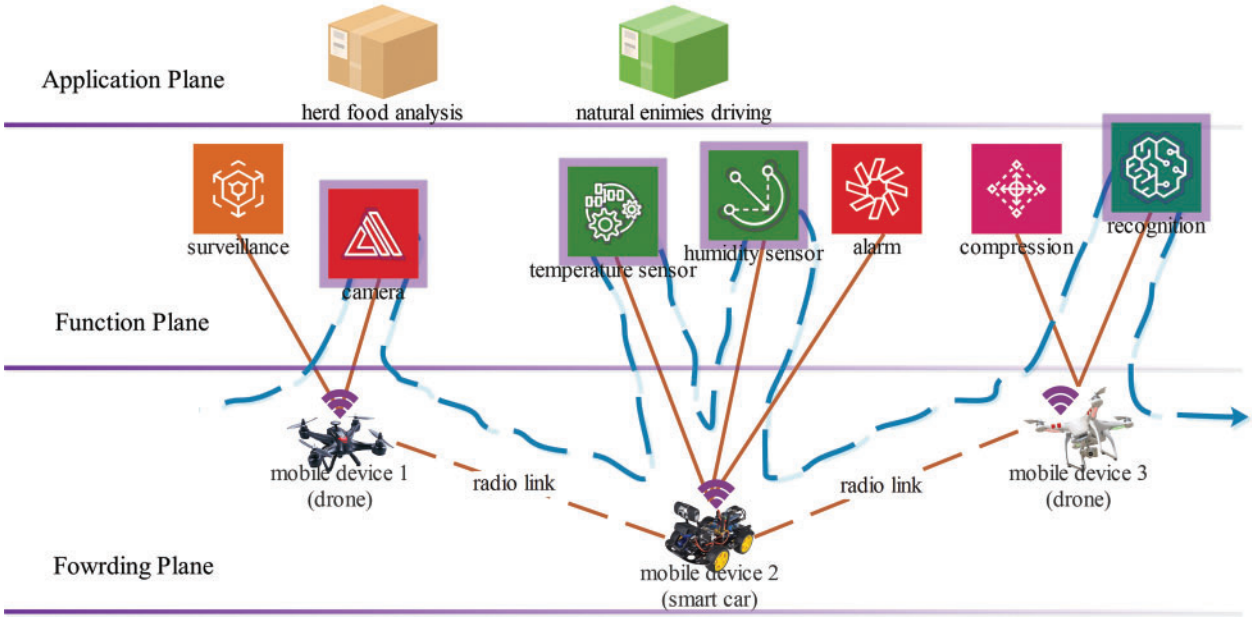

Figure 2: The functional view of the wireless network topology

For example, in Fig. 2, the blue dashed arrow indicates a service function chain that orchestrates camera, temperature sensor, humidity sensor, and recognition as a "herd food analysis" application, across multiple mobile devices, capable of, collectively, taking photos of and recognizing grass species and conditions in the herd's habitat with perceived temperature and humidity status. Also, a surveillance $\rightarrow$ recognition $\rightarrow$ alarm chain can be built as a "natural enemies driving" application that first monitors and recognizes natural enemies, then tries to proactively drive them away with deployed alarming equipment.

Based on the previous discussion, we give the formal definition of the functional mobile wireless network topology in this paper.

Definition 2. The functional mobile wireless network: The functional mobile wireless network is defined as undirected graph $G = (V, F, E)$, where $V = \{v_1, v_2, \cdots, v_{||V||}\}$ represents mobile devices, $F = \cup_{i=1}^{||V||} F_i = \{f_{1,1}, \cdots, f_{i,j}, \cdots, f_{||V||,F||V||}\}$ represents all on-device functions, and $E = \{e_1, e_2, \cdots, e||_E||\}$ represents radio links, where e_k can also be depicted as $e_{i,j}$ if v_i and v_j can communicate directly, and functional links, where e_k can also be depicted as $e_{i,j}$ if v_i has $f_{i,j}$ deployed on it.

Every radio link $e_{i,j}$ will be assigned with a cost according to 3.1. The cost for a functional link is set to 0. According to the previous definition, mobile devices and functions are all equally treated as nodes in an undirected graph, so that graph theories can be applied for service function chaining to orchestrate function-device interleaved chains in a single graph query.

3 Problem Statement and Modeling

3.1 Cost Evaluation

Note that gathered metrics of a mobile device are usually a vector, including bandwidth, delay, packet loss, etc. To simplify orchestration and reduce the problem dimensions, a vector of metrics

can be transformed as a scalar value (i.e., the "cost"), as per the properties of metrics. Let $c_{i,j}$ denote the cost corresponding to $q_{i,j}$, i.e., the j-th metric of mobile device v_i. We consider the following metrics and their transformation methods, so that they can be evaluated as additive costs to apply Dijkstra-like algorithms for shortest paths-based service function chaining: (1) delay, $c_{i,j} = q_{i,j}$; (2) bandwidth, $c_{i,j} = \frac{q_{bench}}{q_{i,j}}$, where q_{bench} is the benchmark bandwidth similar to that in OSPF; (3) packet loss, $c_{i,j} = log\left(\frac{1}{1-q_{i,j}}\right)$; (4) availability, $c_{i,j} = log\left(\frac{1}{q_{i,j}}\right)$. After transformation, every mobile can be evaluated with a cost function using weighted summation: $cost(v_i) = \sum_{j=1}^{||Q_i||} w_j \times c_{i,j}$, where $\sum_{j=1}^{||Q_i||} w_j = 1$, which is then divided equally to be assigned to its effective physical radio links, i.e., $cost(e_k) = \frac{cost(v_i)}{N_i}$, where N_i is the number of radio links e_k of mobile device v_i, to apply Dijkstra-based algorithms for service function chaining.

3.2 The Orchestration Model

Since mobile devices have limited computing capabilities and functionalities, to accomplish complex surveillance tasks, it might require various functions to be orchestrated as a whole application, so that devices collaborate smoothly. That is, a service function chain consists of multiple functions. Considering the instability and volatility of the wireless network, service function chains have higher probability of failure, thus the necessity of finding multiple chains. Meanwhile, chains should also reflect the quality in addition to functions. Based on these intuitions, we model the service function chaining of mobile devices as multi-path multi-function orchestration as follows.

Definition 3. The Multi-Path Multi-Function Orchestration (MPMF): Given an orchestration request $F' = \langle f'_1, f'_2, \ldots, f'_{||F'||} \rangle$ where $f'_i \in F$, and a positive integer K, the MPMF orchestration finds the set $P^K = \{p^1, p^2, \ldots, p^K\} \subseteq P_{F'}$, where $P_{F'}$ is the set containing all the chains from f'_1 to $f'_{||F'||}$, such that:

- $seg_i = \langle f'_i, v'_1, v'_2, \ldots, v'_j, \ldots, f'_{i+1} \rangle$, $v'_j \in V$, $f'_i \in F$, i.e., two consecutive functions are chained by one or more hosting mobile devices.
- $p^k = \langle seg_1, seg_2, \ldots, seg_{f'_{||F'||}} \rangle$, $\forall k \in \{1, 2, \ldots, K\}$, i.e., a chain must traverse all the functions in the order demanded by F'.
- $\forall i, j \in \{1, 2, \ldots, K\}$, $p^i \neq p^j$, i.e., no duplicated chain.
- $\forall k \in \{1, 2, \ldots, K-1\}$, $cost(p^k) \leq cost(p^{k+1})$, i.e., p^k is found before p^{k+1} (shorter chains are found earlier).
- $\forall p \in P_{F'} - P^K$, $cost(p^K) \leq cost(p)$, i.e., only the K least-costly chains are to be found.

4 The Orchestration Algorithm

Intuitively, to solve the MPMF problem is to enumerate all sub-paths seg_i ordered by cost between two consecutive functions f'_i and f'_{i+1}, and concatenate them to pick the K least-costly chains as per the order of orchestration request F'. This is feasible for simpler topologies, but not for more complex topologies with even just tens of mobile devices and a couple of functions per device, since the complexity increases exponentially. Specifically, two major challenges are faced as per the requirements of MPMF. Challenge 1: plain Dijkstra algorithm does not find multiple shortest paths. Besides, the concatenation of sub-paths should also be carefully designed to reduce the problem complexity, other than pure enumeration that could be time-consuming for larger networks. Challenge 2: Dijkstra-like algorithms do not restrain the traversed nodes (i.e., functions

in the case of MPMF), thus not applicable for service function chaining. To this end, we combine and improve Dijkstra and KSP (K Shortest Paths) algorithms to solve MPMF problem that requires the K least-costly chains and traversing all demanded functions.

4.1 The Multi-Path Aspect

We firstly cope with challenge 1, i.e., multi-path aspect. Tabs. 1 and 2 show in detail the adapted and modified KSP algorithm, where dsp(v_s,v_t) calculates the shortest path from the source node v_s to the destination node v_t using Dijkstra algorithm. The routine nextPath(v_s,v_t) finds the next shortest path between v_s and v_t using deviation method (see details in Tab. 2). restoreGraph() is used to set the graph back to its initial linkage states after temporary deletion of arcs. P^K is the ordered set that stores the K shortest chains. P^C is the set that stores the shortest chains candidates. Detailed explanations are shown in the comments in Tabs. 1 and 2.

Table 1: KSP(v_s, v_t, K) algorithm

```
Input:    v_s: the source node
          v_t: the destination node
          K: the number of shortest paths
Output:   P^K: the K shortest paths set
1    P^K = ∅;                                  //P^K: shortest paths set.
2    P^C = ∅;                                  //P^C: shortest path candidates set
3    p^1 = dsp(v_s, v_t);                      //find the shortest path using Dijkstra algorithm
4    if p^1 == NULL then
5        return P^K;                           //no connectivity from source to target
6    end if
7    P^K = P^K ∪ {p^1};
8    while ||P^K|| < K do
9        curSPath = nextPath(v_s, v_t);        //see Tab. 2 to find next shortest
10       if curSPath==NULL then
11           break;                            //no more shortest path, break
12       end if
13   end while
14   return P^K;
```

Table 2: nextPath(v_s, v_t) routine

```
Input:    v_s: the source node
          v_t: the target node
Output:   curSPath: the current shortest path
1   prevSPath = max(P^K);                    //shortest path found in the last iteration
2   for i = 1; i ≤ prevSPath.length; i++ do
3       root = sub_prevSPath(1, i)           //get root for every node
4       for each path from P^K do
5           root_path = sub_path(1, i);
6           if root = root_path then
7               c_{i,i+1} = ∞;               //determine the furthest node as deviation
8           end if
9           spur = dsp(v_i, v_t);            //get spur from deviation node
10          restoreGraph();                  //restore the linkage for deviation nodes
11          if spur=NULL then
12              return NULL;                 //no more valid spur, return NULL
13          end if
14          if root ∩ spur = ∅ then
15              cPath = root+spur;           //construct a loopless candidate
16              P^C = P^C ∪ {cPath};
17          end if
18      end for
19  end for
20  curSPath=min(P^C);                       // shortest from P^C is the current shortest
21  if curSPath != NULL then
22      P^C = P^C − {curSPath}
23      P^K = P^K ∪ {curSPath};
24  end if
25  return curSPath;
```

4.2 The Multi-Function Aspect

We then cope with challenge 2, i.e., multi-function aspect. The MFMP algorithm is shown in Tab. 3. It takes as the input the functions required by the service function chaining request F' which must be traversed. To get the least costly chain that traverses all the demanded functions, the algorithm gets the shortest path for every f'_i and f'_{i+1} function pair along the request F'. These are the shortest segments that constitute the shortest path and are put into the set, P_i^K i.e., the shortest segments between f'_i and f'_{i+1} in the chain. Also, the second shortest segments from f'_i to f'_{i+1} are calculated for later iterations. The above is the initialization of the algorithm, shown in lines 3–7.

Table 3: MFMP(K, F') algorithm

Input: K: *the number of shortest paths*
$\boldsymbol{F'}$: *functions that must be traversed, i.e., service chaining request*
Output: P^K: *the K shortest paths set*

```
1   P^K = ∅;                                          //P^K: shortest paths set.
2   P^C = ∅;                                          //P^C: shortest path candidates set
3   for i = 1; (i + 1) ≤ chain.length; i++ do
4       sPart = dsp(f'_i, f'_{i+1}) → P^K_i;
5       p^1 += sPart;
6       nextPath(f'_i, f'_{i+1}) → P^K_i
7   end for
8   if p^1 == NULL then
9       return P^K;                                   //no connectivity from source to target
10  end if
11  P^K = P^K ∪ {p^1};
12  while ||P^K|| < K do
13      for i = 1; (i + 1) ≤ chain.length; i++ do
14          concatNewPaths(P^K_i) → P^C;
15          let I be index of: min(P^K_i.getLast());
16      end for
17      curPath = min(P^C) → P^K;
18      nextPath(f'_I, f'_I) → P^K_I;
19  end while
20  return P^K;
```

Concatenating all the new segments in P_i^K, P_{i+1}^K, $\cdots$, generates a new batch of feasible paths from f_1' to $f'_{||F'||}$ which are put into P^C. The shortest one from P^C is the next shortest path (shown in lines 14 and 17). The next problem is to determine from which node to generate the next segment so that the next batch paths can be found. Since the algorithm invokes nextPath() routine, it is guaranteed the last one in P_i^K is the longest of currently found shortest segments from f_i' to f'_{i+1}. All the P_i^K.getLast() are compared to pick the shortest segment, then the corresponding v_i is the node to derive new segments, i.e., the deviation node (shown in lines 15 and 18). In this way, the algorithm does not have to derive new segments from every node in the next iteration; it only derives new segment from deviation node, thus time complexity is minimized. The above is the main loop of the algorithm, shown in lines 12–19.

5 Experiments

We conduct service function chaining in a wireless network consists of tens mobile devices, each with 5 functions (commonly seen functions in animal husbandry, e.g., surveillance, sensing, etc.), implemented on the smart cars with Raspberry Pi and Raspbian OS. These mobile devices were networked through the methods proposed in our previous work [11]. The orchestrator, i.e., the MPMF algorithm, and the collector are implemented in its orchestration layer (see Fig. 3).

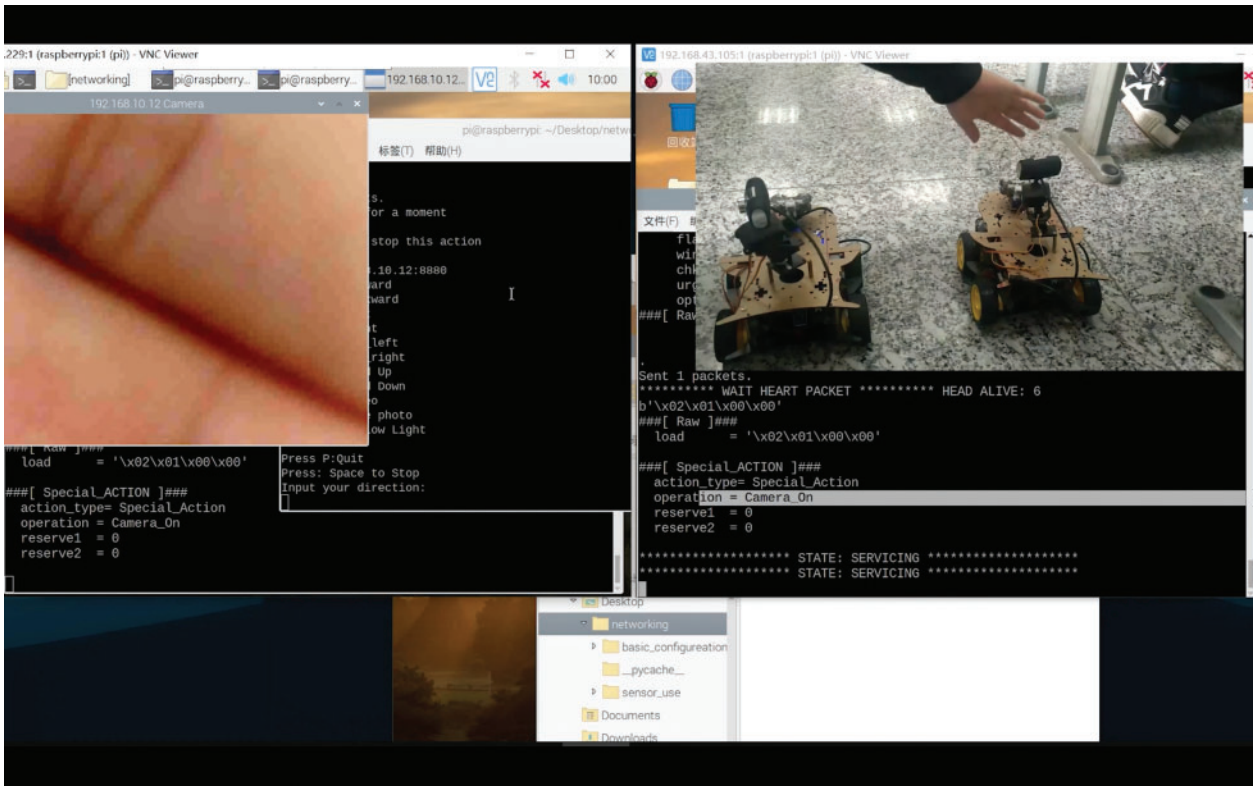

Figure 3: The prototype based on Raspberry Pi and Raspbian OS

Several counterparts capable of service function chaining with necessary modifications, CPSPARQL [12–14] and Virtuoso (discussion on them can be found in Section 6), are compared with MPMF. The experiments are conducted in two wireless networks, containing 10 and 20 mobile devices respectively, and the results are shown in Figs. 4a and 4b, where the suffix "-x" in the legends indicates the number of functions in a service function chaining request, i.e., how many functions to be traversed by a chain. All counterparts are evaluated by chaining time in ms (the y-axis) when finding K different chains (the x-axis). MPMF conducts chaining by means of deviation method. That is, based on the current shortest path, the most appropriate node is selected, from which there derives a new shortest path that traverse all the required functions, instead of enumerating all the possible paths. In this way, the K least-costly paths are found and ordered one by one without enumeration. This, at large, saves chaining time. As we can see from the figure, MPMF scales almost linear to the number of functions demanded by the request and the number of chains to be found. And its performance is time-efficient. For the scenario of 10 mobile devices, it takes about 200 ms for MFMP to find 5 chains that traverse 6 functions, among which the shortest one is used as the primary chain and others stand by as backups in case of any failure or malfunction of the primary one. It takes about 475 ms to accomplish the same task for the 20 mobile devices. See orange lines in Figs. 4a and 4b.

CPSPARQL takes similar philosophy to our work, to compose service function chains, though different in techniques. That is, CPSPARQL finds and orders the K least-costly chains gradually. Notice that the performance of CPSPARQL is less-efficient with regard to time consumed. One major reason for this is that CPSPARQL uses more redundant data structures of RDF (Resource

Description Framework) triples because the primary usage of CPSPARQL is in the field of Semantic Web, where, usually, data are stored as half-structured triples. Triples take the form of three ordered entities, i.e., [subject → predicate → object], where edge-similar predicates are also modelled as nodes, increasing the size of abstract graph. Expressive enough for triples though, to express a simple relationship using triples, it adds more nodes and edges in the abstract graph, which as a result lowers the performance with regard to time complexity, especially when applied in the orchestration of more complex wireless networks. Therefore, RDF triples might be heavyweight in this scenario. It takes about 354 ms for CPSPARQL to find five 6-function chains in the scenario of 10 mobile devices, and 980 ms in the scenario of 20 mobile devices. See purple lines in Fig. 4a and 4b.

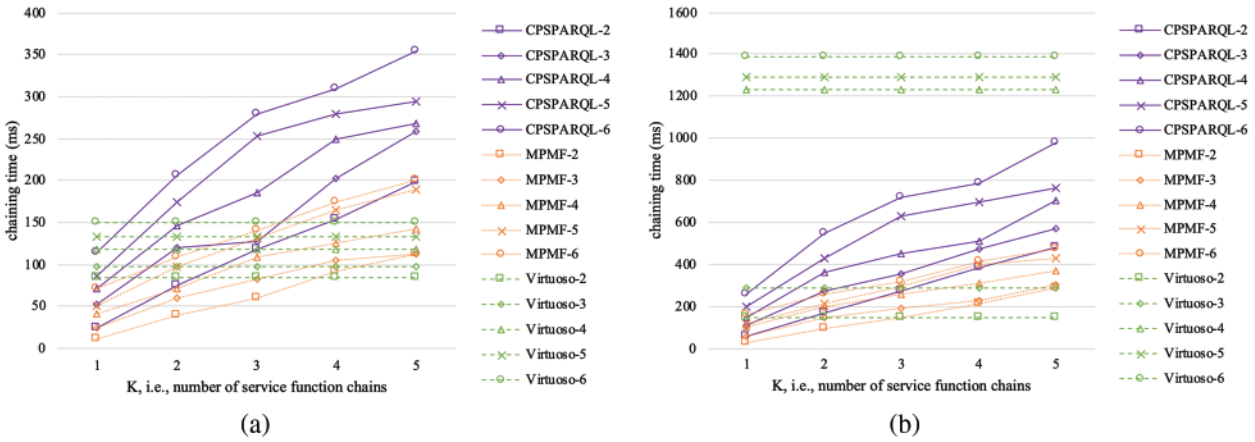

Figure 4: The experiment results (a) a wireless network with 10 mobile devices (b) a wireless network with 20 mobile devices

Virtuoso is also a high-efficient Semantic Web query engine that provides good performance to search useful information in RDF triples. Virtuoso takes a different technical path that enumerates all possible chains that traverse all required functions, and order them in one shot. This is feasible when the network size is small. We can see from Fig. 4a that, Virtuoso, with good optimization of implementation, even outperforms MPMF and CPSPARQL in the scenario of 10 mobile devices, although seemingly time-consuming enumeration strategy is taken. That is, it takes only about 150 ms to find five 6-function chains, superior to MPMF and CPSAPRQL. One thing worth noticing is that Virtuoso provides the same temporal performance for different K (i.e., the number of service function chains) because it enumerates all chains. That is, K is irrelative for Virtuoso. However, in the scenario of 20 mobile devices, the performance of enumeration degrades considerably. To accomplish the same task, it takes about 1385 ms. See green lines in Figs. 4a and 4b.

To summarize, MPMF provides good performance for service function chaining that orchestrates various functions deployed on mobile devices. Therefore, this algorithm is a fitting candidate for the potential usage in the mobile surveillance of animal husbandry in ethnic areas.

6 Related Works

Service function chaining is an active research area in recent years. Reference [15] proposes the low-latency and resource-efficient service function chaining orchestration in network function virtualization (NFV) [16]. However, it does not suit well in the context of wireless orchestration, nor does it find multiple chains to as backups in case the primary one fails in a volatile environment. StEEIRNG [17] tries to arrange a path traversing specific middleboxes by extending the OpenFlow and NOX controller, suitable for service function chaining in the context of SDN (Software-Defined Networking) [18,19]. The key of the extension is the split of a monolithic flow table into several micro tables to constrain the "rule explosion". StEEIRNG solve the path planning problem using Graph Theory. The main deficiency of StEEIRNG is the lack of QoS support. It conducts the chaining mainly based on middleboxes functions. SlickFlow [20] focuses on the source routing (similar to service function chaining) based fault recovery. The fault recovery application on the controller pushes the path information as special headers on the ingress switches. The path consists of several segments where next hop and the alternative path from that hop are contained in each segment. Upon network failure at a certain node, alternative path is adopted proactively by the current node to reduce controller intervention. Reference [21] considers the service function chain orchestration in a network slice across multiple domains. Reference [22] also considers a similar energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks, which is formulated as an integer linear programming (ILP) model to find an optimal solution. A low-complexity heuristic algorithm is derived to chain VNFs across multiple domains.

We can see that recent works in service function chaining mainly focus on the wired network scenarios, as opposed to wireless networks. Therefore, these works do not fit well in the context of wireless orchestration with node mobility that increases the instability and volatility. Our work differs from these works in that we consider multiple backup chains in case of chain failure due to various factors in wireless networks.

CPSPARQL and Virtuoso, although they originate from the Semantic Web field, can be applied for chaining purposes with necessary modifications and adaptations. Nevertheless, their performances are not as good as expected in mobile device orchestration, due to their RDF-based data structures (i.e., triple-like) contains redundant nodes and edges that compromise the performance in case of more complex networks.

7 Conclusion

To implement mobile surveillance for animal husbandry in ethic areas of China, where communication or networking infrastructure is less-developed, this paper proposes the multi-path service function chaining, which orchestrates multiple chains with the least costly one as the primary chain and several others as backups, to accommodate wireless volatility and instability. The proposed MPMF algorithm performs well in the orchestration in wireless networks with tens of mobile devices.

In this paper, the quality of chains is evaluated by a scalar, i.e., the cost. It can be understood as a "best effort" service, since no quality constraints are imposed [23], i.e., no restriction on delay, bandwidth, packet loss, etc. If multiple constraints in addition to quality preference are put forward for service function chaining, the complexity becomes high. We will study the possibility of applying reinforcement learning in the field of wireless service function chaining in the future

work, to provide an efficient and topology-adaptive service function chaining framework with self-learning capabilities.

Acknowledgement: The authors would like to thank the time and efforts by the editors and reviewers.

Funding Statement: This research was partially supported by the National Key Research and Development Program of China (2018YFC1507005), China Postdoctoral Science Foundation (2018M643448), Sichuan Science and Technology Program (2020YFG0189), and Fundamental Research Funds for the Central Universities, Southwest Minzu University (2020NQN18).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

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

[2] A. Shahraki, A. Taherkordi, Ø. Haugen and F. Eliassen, "A survey and future directions on clustering: From WSNs to IoT and modern networking paradigms," *IEEE Transactions on Network and Service Management,* vol. 18, no. 2, pp. 2242–2274, 2021.

[3] Z. Li, B. Chang, S. Wang, A. Liu, F. Zeng *et al.,* "Dynamic compressive wide-band spectrum sensing based on channel energy reconstruction in cognitive internet of things," *IEEE Transactions on Industrial Informatics,* vol. 14, no. 6, pp. 2598–2607, 2018.

[4] R. Lohiya and A. Thakkar, "Application domains, evaluation data sets, and research challenges of IoT: A systematic review," *IEEE Internet of Things Journal,* vol. 8, no. 11, pp. 8774–8798, 2021.

[5] H. Zhu, D. Gao and S. Zhang, "A perceptron algorithm for forest fire prediction based on wireless sensor networks," *Journal on Internet of Things,* vol. 1, no. 1, pp. 25–31, 2019.

[6] D. Corral-Plaza, J. Boubeta-Puig, G. Ortiz and A. Garcia-de-Prado, "An internet of things platform for air station remote sensing and smart monitoring," *Computer Systems Science and Engineering,* vol. 35, no. 1, pp. 5–12, 2020.

[7] S. Jha, L. Nkenyereye, G. P. Joshi and E. Yang, "Mitigating and monitoring smart city using internet of things," *Computers, Materials & Continua,* vol. 65, no. 2, pp. 1059–1079, 2020.

[8] W. He, S. Guo, Y. Liang, R. Ma, X. Qiu *et al.,* "Qos-aware and resource-efficient dynamic slicing mechanism for internet of things," *Computers, Materials & Continua,* vol. 61, no. 3, pp. 1345–1364, 2019.

[9] Z. Ye, X. Cao, J. Wang, H. Yu and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Network,* vol. 30, no. 3, pp. 81–87, 2016.

[10] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci *et al.,* "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine,* vol. 55, no. 2, pp. 216–223, 2017.

[11] X. Chen, T. Wu, G. Sun and H. Yu, "Software-defined MANET swarm for mobile monitoring in hydropower plants," *IEEE Access,* vol. 7, no. 1, pp. 152243–152257, 2019.

[12] F. Alkhateeb, J.-F. Baget and J. Euzenat, "Extending SPARQL with regular expression patterns (for querying RDF)," *Web Semantics,* vol. 7, no. 1, pp. 57–73, 2009.

[13] F. Alkhateeb and J. Euzenat, "Constrained regular expressions for answering RDF-path queries modulo RDFS," *International Journal of Web Information Systems,* vol. 10, no. 1, pp. 24–50, 2014.

[14] F. Alkhateeb, J.-F. Baget and J. Euzenat, "Constrained regular expressions in SPARQL," in *Proc. Int. Conf. on Semantic Web and Web Services (SWWS)*, Las Vegas, Nevada, USA, pp. 91–99, 2008.

[15] G. Sun, Z. Xu, H. Yu, X. Chen, V. Chang *et al.,* "Low-latency and resource-efficient service function chaining orchestration in network function virtualization," *IEEE Internet of Things Journal,* vol. 7, no. 7, pp. 5760–5772, 2020.

[16] A. Alsarhan, A. Itradat, A. Y. Al-Dubai, A. Y. Zomaya and G. Min, "Adaptive resource allocation and provisioning in multi-service cloud environments," *IEEE Transactions on Parallel and Distributed Systems,* vol. 29, no. 1, pp. 31–42, 2018.

[17] Y. Zhang, N. Beheshti, L. Beliveau and G. Lefebvre, "StEERING: A software-defined networking for inline service chaining," in *Proc. IEEE Int. Conf. on Network Protocols*, Göttingen, Germany, pp. 1–10, 2013.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson *et al.,* "Openflow: Enabling innovation in campus networks," *SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69–74, 2008.

[19] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky *et al.,* "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[20] R. M. Ramos, M. Martinello and C. E. Rothenberg, "Slickflow: Resilient source routing in data center networks unlocked by openFlow," in *Proc. IEEE Conf. on Local Computer Networks (LCN)*, Sydney, Australia, pp. 606–613, 2013.

[21] G. Sun, Y. Li, D. Liao and V. Chang, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," *IEEE Transactions on Network and Service Management,* vol. 15, no. 3, pp. 1175–1191, 2018.

[22] G. Sun, Y. Li, H. Yu, A. V. Vasilakos, X. Du *et al.,* "Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks," *Future Generation Computer Systems,* vol. 91, no. 1, pp. 347–360, 2019.

[23] S. Long, W. Long, Z. Li, K. Li, Y. Xia *et al.,* "A game-based approach for cost-aware task assignment with QoS constraint in collaborative edge and cloud environments," *IEEE Transactions on Parallel and Distributed Systems,* vol. 32, no. 7, pp. 1629–1640, 2021.