sensors

MDPI

Article

The SDN Approach for the Aggregation/Disaggregation of Sensor Data

Yi-Bing Lin *, Shie-Yuan Wang, Ching-Chun Huang and Chia-Ming Wu

Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan; shieyuan@cs.nctu.edu.tw (S.-Y.W.); d3350233.cs05g@nctu.edu.tw (C.-C.H.); pcmwu.g@gmail.com (C.-M.W.)

* Correspondence: liny@csie.nctu.edu.tw; Tel.: +886-3-513-1500

Received: 23 May 2018; Accepted: 21 June 2018; Published: 25 June 2018

Abstract: In many Internet of Things (IoT) applications, large numbers of small sensor data are delivered in the network, which may cause heavy traffics. To reduce the number of messages delivered from the sensor devices to the IoT server, a promising approach is to aggregate several small IoT messages into a large packet before they are delivered through the network. When the packets arrive at the destination, they are disaggregated into the original IoT messages. In the existing solutions, packet aggregation/disaggregation is performed by software at the server, which results in long delays and low throughputs. To resolve the above issue, this paper utilizes the programmable Software Defined Networking (SDN) switch to program quick packet aggregation and disaggregation. Specifically, we consider the Programming Protocol-Independent Packet Processor (P4) technology. We design and develop novel P4 programs for aggregation and disaggregation in commercial P4 switches. Our study indicates that packet aggregation can be achieved in a P4 switch with its line rate (without extra packet processing cost). On the other hand, to disaggregate a packet that combines N IoT messages, the processing time is about the same as processing N individual IoT messages. Our implementation conducts IoT message aggregation at the highest bit rate (100 Gbps) that has not been found in the literature. We further propose to provide a small buffer in the P4 switch to significantly reduce the processing power for disaggregating a packet.

Keywords: aggregation; disaggregation; Internet of Things; programmable switch; P4; sensor data; Software Defined Networking

1. Introduction

Many Internet of Things (IoT) technologies have been used in applications for people flow, traffic flow, logistics flow, money flow, smart home, interactive art design, and so on. To promote IoT applications on campus, National Chiao Tung University (NCTU) is deploying several smart campus applications. These applications are created based on IoTtalk [1–4], an IoT application management platform that can be installed on top of IoT protocols such as AllJoyn [5], OM2M [6], OpenMTC [7], and an arbitrary proprietary protocol.

IoTtalk allows a designer to quickly establish connections and meaningful interactions between IoT devices. Figure 1 illustrates the simplified IoTtalk network architecture. In this client-server architecture, the IoT devices (the clients) are connected to the IoTtalk engine (the server; Figure 1(1)) in the Internet through wireline or wireless technologies. When an IoT device is connected to the IoTtalk server, the server automatically creates the network application (Figure 1(2)) for the device to interact with other devices. The IoT device can be a group of sensors (such as color sensor and temperature sensor) or controllers (such as switches and buttons), which is called an input device (Figure 1(3)). The device is installed as a device application (DA; Figure 1(4)) that collects the sensor data generated by the sensor/controller of the device. The DA then sends the sensor data to the IoTtalk

server for processing (see path (4) → (2) in Figure 1). Similarly, an IoT device is called an output device (Figure 1(5)) if it is a group of actuators (e.g., robot arm). The DA of the output device receives data from the IoTtalk engine (see path (2) → (6) in Figure 1) to drive its actuators.

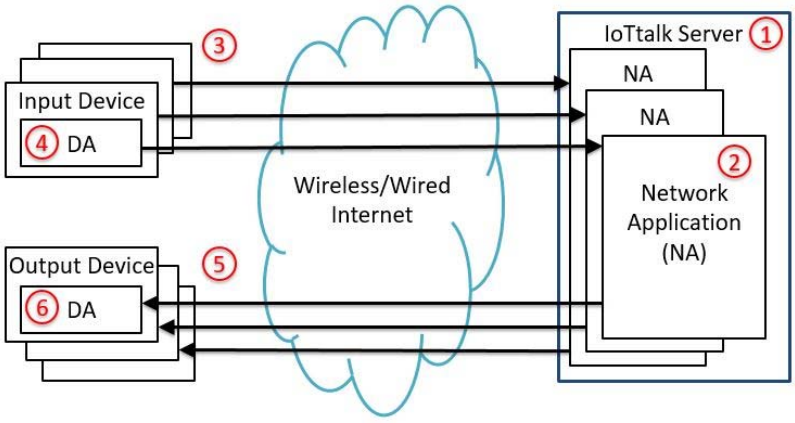

Figure 1. Simplified IoTtalk network architecture.

Based on IoTtalk, we have developed outdoor applications using LoRA-based PM2.5 detection [8] and NB-IoT-based parking (Figure 2), an emergency button, and dog tracking [9]. Besides outdoor applications, we have also created indoor IoT applications at the guest houses, the student dorms, and a research laboratory called Room 311 in the MIRC building (Figure 3). More than 100 sensors and actuators are installed in Room 311, and data are collected every day for machine-learning analysis. In addition, the smart home appliances are being installed in the newly built Graduate Dormitory 3. This student dorm will accommodate 1200 graduate students. A large amount of data (IoT messages) have been collected in Room 311, MIRC, and Graduate Dormitory 3. These IoT messages are UDP packets with small payloads. For example, inside an elevator car, we have installed a 3D accelerometer, a barometer, and a thermometer. Outside the car, the motor is attached to a biaxial vibration accelerator. The volume of data produced by these sensors is about 12 MB/minute. There are about 100 elevator cars on campus, which produce data at the speed of 20 MB/second. There are over 20 IoT applications exercised in NCTU, and these applications generate large volumes of small-packet data through path (4) → (2) in Figure 1.

Figure 2. Smart parking in NCTU campus.

To reduce the number of sensor data (IoT message) delivered from the IoT devices to the IoT server, a promising approach is to aggregate several small IoT messages into a large packet before they are delivered through the network [10]. When the packets arrive at the destination (in our example, the IoTtalk server is installed in the NCTU's private cloud at the computer center), they are disaggregated into the original IoT messages. In this paper, we show how IoT data can be effectively aggregated and disaggregated. Specifically, we consider the software defined networking (SDN) approach to design IoT data aggregation and disaggregation for IoTtalk.

Figure 3. Room 311 in the MIRC building.

The paper is organized as follows. Section 2 introduces the programmable SDN switches. Section 3 describes how programmable SDN switches can be utilized for sensor data aggregation and disaggregation. Section 4 shows how to improve disaggregation performance for the SDN switches. Section 5 compares our solution with the previous studies. Specifically, we show an aggregation solution based on SDN controllers. Finally, Section 6 concludes our work and points out a future research direction.

2. Programmable SDN Switch

To resolve the sensor data delivery issue, Software Defined Networking (SDN) is considered as a promising solution. SDN decouples the data and the control planes [11]. Recently, programmability of the data plane has become one of the most desirable SDN features, which allows the user to describe how to process the packets in an SDN switch. This task can be achieved by using tools such as Programming Protocol-Independent Packet Processor (P4). P4 is a reconfigurable, multi-platform, protocol and target-independent packet processing language, which is used to facilitate dynamically programmable and extensible packet processing in the SDN data plane [12–14]. In SDN, a switch uses a set of "match+action" flow tables to apply rules for packet processing, and P4 provides an efficient way to configure the packet processing pipelines to achieve this goal. In a typical communications network, a packet consists of a packet header and the payload, and the header includes several fields defined by the network protocol. A P4 program describes how packet headers are parsed and their fields are processed by using the flow tables, in which the matched operations may modify the header fields of packets or the content of the metadata registers.

Figure 4 illustrates the P4 abstract forwarding model elaborated in [15] (the conference paper version of this paper), and the details are reiterated here for the reader's benefit. In this model, a configuration file describes all components declared in the P4 program, including the parse graph (Figure 4(1)), the table configuration (Figure 4(2)), and an imperative control program that defines the order of the tables to be applied in the processing flow (Figure 4(3)). This configuration file is loaded into the switch to specify the parser (Figure 4(4)) followed by the flow tables in the ingress and the egress pipelines (Figure 4(5),(6)). The ingress pipeline generates an egress specification that determines the set of ports (and number of packet instances for each port) to which the packet will be sent. Between these two pipelines, there is a traffic manager (TM; see Figure 4(7)) that performs the queueing functions for the packets. At the TM, the packets are queued before being sent to the egress pipeline, through which the packet header may be further modified. At the deparser (Figure 4(8)), the headers are assembled back to a well-formed packet.

The parser extracts the header fields and the payload of an incoming packet following the parse graph defined in the P4 program. In P4, the parser state transitions are driven by the values of the fields in the headers.

Based on the above P4 forwarding model, we propose a novel approach that utilizes the header manipulation of the P4 switch to aggregate small packets into large packets and disaggregate large packets back into small packets. We describe our approach through the software switch behavior model bmv2, and actually implemented our algorithm in two EdgeCore P4 switches, which use barefoot's Tofino P4 chips. The bmv2 framework enables the developers to implement P4 applications

on a software switch. This framework is the de-facto architecture for most developers, as it is roughly equivalent to the abstract forwarding model illustrated in Figure 4. The Tofino chip is based on the Protocol Independent Switch Architecture (PISA) and can be programmed using P4. With the Capilano Software Development Environment (SDE) P4 compiler toolchain, the developers can compile and run P4 programs at a Tofino P4 switch at the line rate up to 100 Gbps per port. There are 64 ports in a P4 chip, which sum up to 6.5 Tbps.

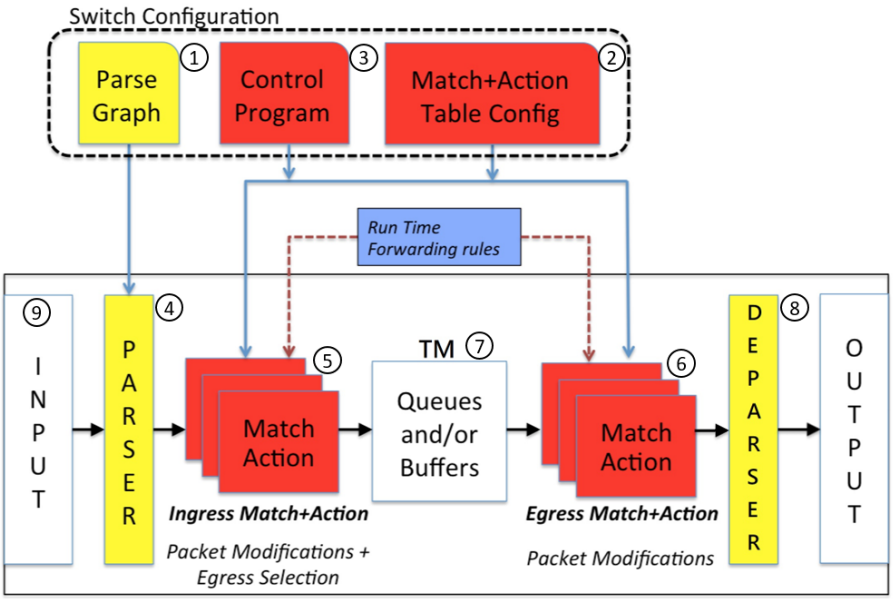

Figure 4. The P4 abstract forwarding model.

3. Performance of IoT Packet Aggregation and Disaggregation in the P4 Switch

We have integrated IoTtalk in the SDN environment. The architecture will be illustrated in Section 5 later. In this section, Figure 5 illustrates a simplified general network architecture for IoT packet aggregation and disaggregation. In this architecture, K IoT devices (Figure 5(1)) send small packets (the IoT messages) to the IoT server (Figure 5(4)) though the P4 switches (Figure 5(2),(3)). When N consecutive IoT messages arrive at the first P4 switch, they are aggregated into a large packet. After aggregation, the first P4 switch sends the aggregated packet to the second P4 switch. The second P4 switch then disaggregates the packet into the original IoT messages, and sends them to the IoT server. Between these two P4 switches, there may be a wide area network not shown in the figure.

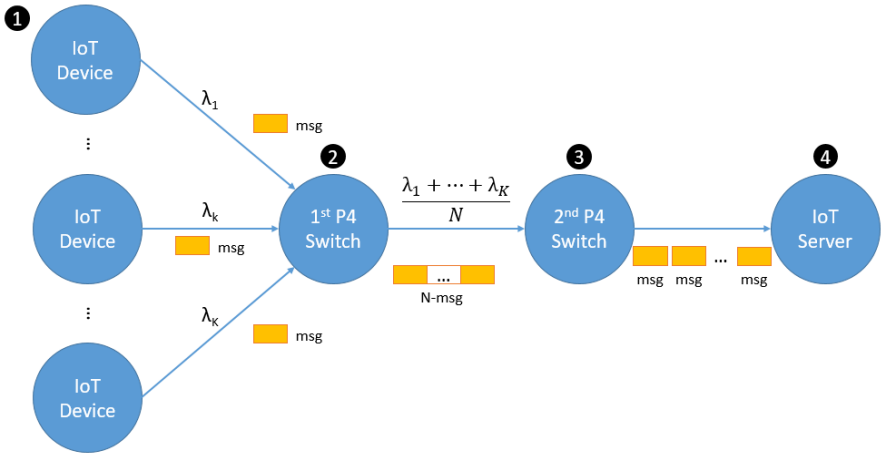

Figure 5. The P4 switch network architecture for IoT packet aggregation and disaggregation.

In our approach, the first P4 switch aggregates N IoT messages into one packet. Denote the aggregated packet as an N-packet. Figure 6 shows the packet formats of an IoT message and an N-packet for $N = 8$. For illustration purposes, we assume that an IoT message (Figure 6a) is a UDP packet with a 16-byte payload. In our design, the 16-byte payload is treated as a header called msg (Figure 6b). Between the UDP header and the payload is a 6-byte flag header (Figure 6c), of which the "type" field is used to indicate if the packet is an IoT message or an N-packet. The flag header is declared as

```
header_type flag_t {
    fields {
        type: 8;
        padding: 40;
    }
}
header flag_t flag;
```

Note that the flag header has a 40-bit "padding" field to make the minimum length of an Ethernet frame 64 bytes. To meet this requirement, we purposely pad the length of the flag header to be 6 bytes so that together with a 14-byte Ethernet header, a 20-byte IP header, an 8-byte UDP header, and a 16-byte msg, the total length of an IoT message is 64 bytes. For an N-packet (Figure 6d), we consider the payloads aggregated from the N consecutive IoT messages as a header called agg (Figure 6e). That is, the n-th header field msg_n is the payload of the n-th IoT message (in this example, $1 \leq n \leq N = 8$). Therefore, the agg header is declared as

```
header_type agg_msg_t {
    fields {
        msg1: 128;
        msg2: 128;
        msg3: 128;
        msg4: 128;
        msg5: 128;
        msg6: 128;
        msg7: 128;
        msg8: 128;
    }
}
header agg_msg_t agg;
```

We treat the payloads of a packet as a header and showed how to use the registers and the pipelines of the first P4 switch to conduct header operations to transform the payloads of N IoT messages (Figure 6b) into the payload of an aggregated packet (Figure 6e). When an IoT message arrives, the payload is parsed as a header. At Figure 4(4), the parser processes the Ethernet, the IP, and the UDP header (Figure 6f). When the flag_t header (Figure 6c) is encountered, the parser checks the value of the "type" field, then the parser extracts either the msg header (if "type" is 0xfa) or the agg header (if "type" is 0xfc). The P4 switch checks if the incoming packet is an IoT message (i.e., whether it has a valid msg header). If so, the payload of the message is considered as a header field that is saved in the register. The IoT message is then dropped. The P4 switch continues to collect the payloads of the incoming IoT messages in the register. When N payloads have been collected, these saved payloads are considered as header fields to produce the aggregated payload (Figure 6e). The aggregated packet is then sent out of the switch.

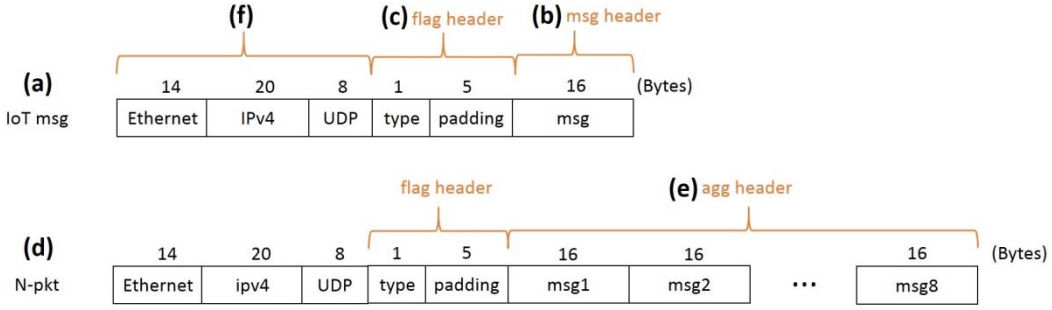

Figure 6. The packet formats of an IoT message and an N-packet (N = 8).

In Figure 5, for $1 \leq k \leq K$, device k generates the packets following an arbitrary process with the rate λ_k. Then, the net arrivals of the packets to the first P4 switch form a Poisson process with the arrival rate $\lambda = \sum_{k=1}^{K} \lambda_k$. Note that the Poisson property of merged input streams is observed in the multiple sources of IoT traffic in Room 311, which is consistent with the superposition property of the Poisson process [16]. We first show that the inter-arrival times of the N-packets arriving at the second P4 switch follow an N-stage Erlang distribution with the mean N/λ:

$$f_N(t_N) = \frac{\lambda^N {t_N}^{N-1} e^{-\lambda t_N}}{(N-1)!}$$

and its Laplace transform is

$$f_N^*(s) = \int_{t_N=0}^{\infty} f_N(t_N) e^{-st_N} dt_N = \left(\frac{\lambda}{s+\lambda}\right)^N \quad (1)$$

Based on the Poisson property of merged input sources, we have conducted discrete event simulation to model the packet departures of the first P4 switch to see if the packet departures (i.e., the packet arrivals of the second P4 switch) follow the Erlang-N distribution. The first P4 switch is represented by either an M/M/1 or an M/G/1 queue in the simulation. Figure 7 compares the simulation results (the histogram bars) against the Erlang density function (the curves) in which the service times of the P4 switch are exponentially distributed (Figure 7a–c) or fixed (Figure 7d). In Figure 7a, the service rate $\mu = 1.1\lambda$ and N = 5. The figure indicates that the departure process does fit Erlang-5 distribution. If we increase the service rate μ from 1.1λ to 2.2λ (Figure 7b) or increase the aggregation factor N from 5 to 8 (Figure 7c), the departure process fits the Erlang distribution better. Also, if the service times are fixed, then the departure fits the Erlang distribution better (Figure 7d). We note that for same kind of packets (e.g., the IoT messages in this paper), the service times are fixed in a P4 switch.

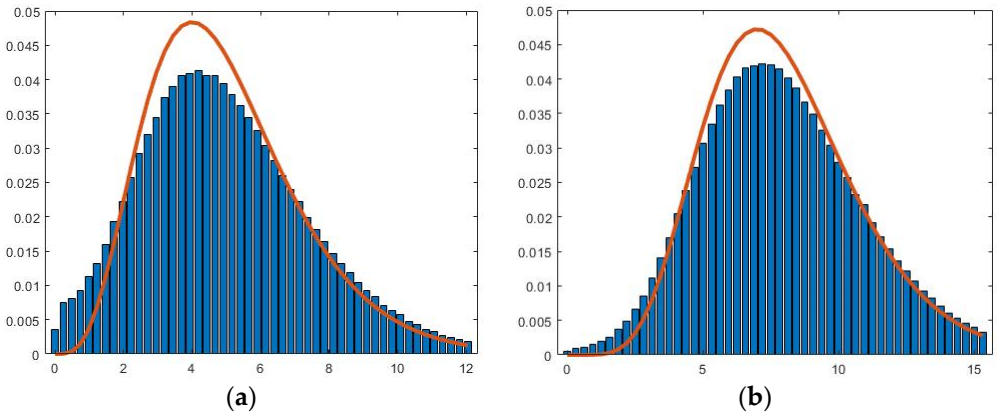

Figure 7. *Cont.*

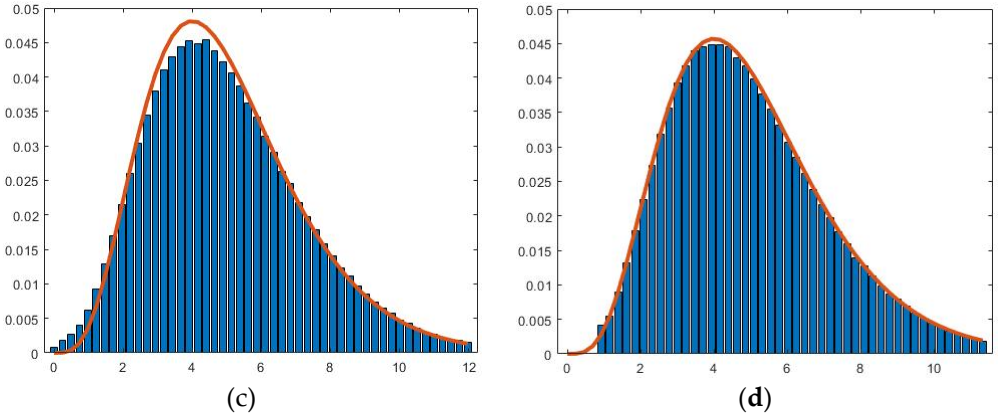

Figure 7. Validation of the Erlang departure process of the first P4 switch: (**a**) exponential ($\mu = 1.1\lambda$, $N = 5$); (**b**) exponential ($\mu = 1.1\lambda$, $N = 8$); (**c**) exponential ($\mu = 2.2\lambda$, $N = 5$); and (**d**) fixed ($\mu = 1.1\lambda$, $N = 5$).

The switches typically handle the packets without queueing, and its maximum processing capability per output port is given by its "line rate". When the packets arrive within the line rate, no packet is dropped at the switch. We have actually implemented our algorithm in the EdgeCore P4 switches based on Barefoot Tofino chip (see Figure 8a).

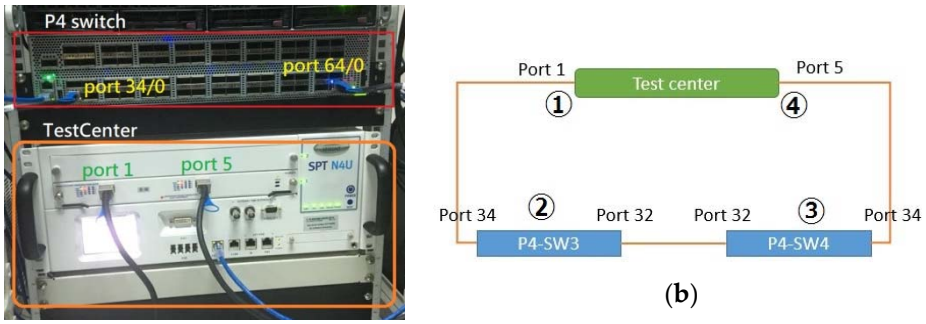

Figure 8. The experimental environment with 2 EdgeCore P4 switches and the TestCenter: (**a**) physical connection of the EdgeCore switch and the TestCenter and (**b**) functional block diagram.

Our study shows that for any N value, aggregation of the N-packets does not slow down the line rate of the first P4 switch (Figure 8b(2)). However, packet disaggregation at the second P4 switch (Figure 8b(3)) slows down packet processing of the switch. This phenomenon is due to the fact that the pipeline architecture is used, and to disaggregate an N-packet requires going through the ingress pipeline for N times.

On the other hand, the packet header overhead decreases as N increases. Specifically, for the example in Figure 6, the size of an N-packet is

$$S_N = 48 + 16N \tag{2}$$

in which 48 bytes come from the Ethernet (14 bytes), the IPv4 (20 bytes), the UDP (8 bytes), and the flag (6 bytes) headers, and every agg header contributes 16 bytes (Figure 6). From (2), the payload to packet ratio

$$p_N = \frac{16N}{S_N} = \frac{N}{4+N} \tag{3}$$

In (3), p_N is the ratio of the useful information delivered through the switch. If the throughput of a P4 switch is X, then we can define the goodput as

$$X_N = p_N X \text{ for } N \geq 1 \tag{4}$$

The network architecture in Figure 5 is actually implemented in Figure 8, in which two EdgeCore P4 switches are connected to serve as P4 switches 1 and 2 in Figure 8b(2),(3). The Spirent TestCenter [17] provides the traffic sources (Figure 8b(1)) to pump the combined IoT message traffic to the first EdgeCore P4 switch. Spirent TestCenter is an end-to-end testing solution that delivers high performance with deterministic answers. We use it to serve as the 100 Gbps traffic source, which is the highest speed in the world. The TestCenter also serves as the IoT server (Figure 8(4)), which receives the packets disaggregated at the second EdgeCore P4 switch. With the Spirent TestCenter, we are able to pump up to 100 Gbps line rate of packets in the experiments.

Based on the throughput measurements X of our experiments, Figure 9 shows the goodput X_N against the net packet arrival rate λ from the IoT devices (Figure 5(1)) for different N values. The figure indicates that as N increases, the goodput increases. However, the X_N saturates as λ increases. For a large N, X_N saturates faster. This phenomenon is due to the hardware constraint of the P4 switch. The line rate for a port of the switch is 100 Gbps.

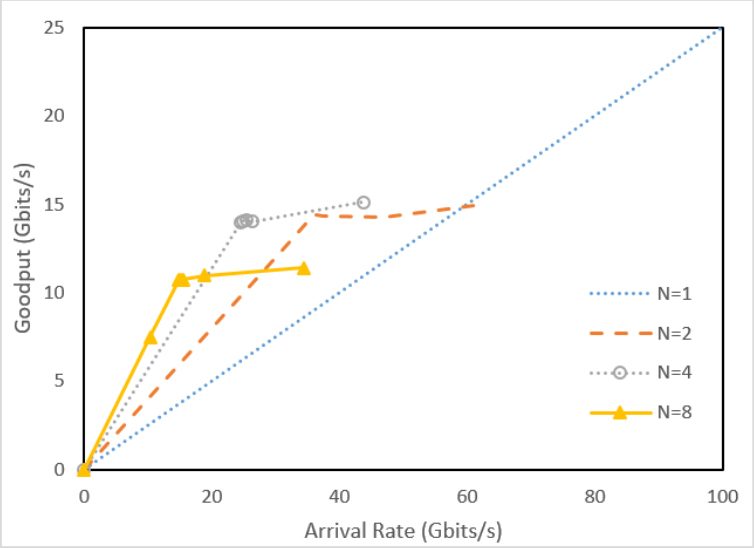

Figure 9. Goodput vs. packet arrival rates for the P4 switch configuration ($1 \leq N \leq 8$).

The service rate μ_N for disaggregating an N-packet cannot be directly measured from the P4 switch. Instead, by using the arrival rate and the goodput in Figure 9, we can use the G/D/1/1 queue to model the P4 switch to derive the disaggregation processing time for an N-packet. We use the goodput when there is no packet loss and without buffering as the service rate. Observing from the measurements of packet disaggregation of the P4 switch implementing our mechanism, we have

$$\frac{1}{\mu_N} \approx \frac{\alpha_N}{\mu_1} \tag{5}$$

in which $0.85 \leq \alpha_N \leq 1.14$. In other words, the processing time for disaggregating an N-packet is roughly the same as that for processing N IoT messages.

4. Packet Disaggregation in the P4 Switch with Queueing

This section shows how the queueing mechanism can effectively reduce the computing power of the P4 switch required to perform disaggregation of the IoT packets.

The previous section considers the case where the switch is operated without the input buffers, which is a typical design of switch. However, in our design for the disaggregation process, an N-packet needs to be looped back to the input port N times through the pipeline (Figure 4(5),(6)). Each time, an IoT message is extracted from the N-packet and sent out of the second P4 switch (Figure 4(8)). Due to this design, the resulting data rate to an input port is N times of the data rate of the N-packets. If the resulting data rate is greater than the line rate of the input port during some time intervals and there is no buffer in the input port, the looped N-packets will have a high chance of being dropped. In contrast, if some buffers can be provided in the input port, then packet drops during the disaggregation process can be reduced. With buffers provided at the input port, some queueing effects at the input port are observed as follows.

Based on the queueing theory, the second P4 switch can be designed to accommodate the traffic $\rho_N = \lambda/(N\mu_N)$ if there is no queueing effect when the packets arrive at the rate λ/N. In the real world, the queueing effect cannot be avoided at the switch if the packets arrive at random. Therefore, we say that the second P4 switch can accommodate traffic ρ_N if the expected waiting time $E[t_W] = \beta/\mu_N$ for a small β value (e.g., $\beta = 0.1$).

Denote the service rate (the line rate) of the second P4 switch as μ_1. If the packets are not aggregated, then the packet arrivals to the second P4 switch are a Poisson process with the rate λ. If the packets are aggregated into N-packets, then the packet arrival rate to the second switch is λ/N, and the packet processing rate μ_N is expressed in (5). With a small buffer time β/μ_N or the buffer size $\lceil\beta\rceil$, we expected a smaller μ_N to keep the same goodput as that for μ_1, and therefore the saved computing power can be allocated for other P4 traffics.

To validate the analytic and the simulation models, let the processing time of the second P4 switch for an N-packet have an exponential distribution with the rate μ_N. After we have validated the simulation model, the exponential assumption is relaxed to accommodate fixed processing time that fits the real P4 switch operation. The μ_N value obtained in the previous section is used. The study in the previous section indicates that the inter-arrival times of the N-packets to the second P4 switch have the Erlang-N distribution with the rate λ/N. Then, the second P4 switch can be modeled as an $E_N/M/1$ queue by assuming that the packet processing time for an N-packet is exponentially distributed. This model is used to validate the simulation experiments. Then, we use the validated simulation to investigate the configuration in Figure 5 (i.e., Figure 8b) with the fixed processing times in the second EdgeCore P4 switch. Let t_R and t_W be the response and the waiting times of an N-packet at the second P4 switch, respectively. Then, from [16] we have

$$E[t_W] = \frac{x}{\mu_N(1-x)} \text{ and } E[t_R] = E[t_W] + \frac{1}{\mu_N} = \frac{1}{\mu_N(1-x)} \tag{6}$$

From (1), x is expressed as

$$x = f_N^*(\mu_N(1-x)) = \left[\frac{\lambda}{\mu_N(1-x)+\lambda}\right]^N = \left(\frac{\theta}{1-x+\theta}\right)^N \tag{7}$$

and $\theta = \lambda/\mu_N$. For N = 2, (7) is solved to yield

$$x = \frac{1+2\theta-\sqrt{1+4\theta}}{2} = \frac{\mu_2+2\lambda-\sqrt{\mu_2(\mu_2+4\lambda)}}{2\mu_2} \tag{8}$$

Substitute (8) to (6) to yield

$$E[t_W] = \frac{\frac{\mu_2+2\lambda-\sqrt{\mu_2(\mu_2+4\lambda)}}{2\mu_2}}{\mu_2\left[1-\frac{\mu_2+2\lambda-\sqrt{\mu_2(\mu_2+4\lambda)}}{2\mu_2}\right]} = \frac{\mu_2+2\lambda-\sqrt{\mu_2(\mu_2+4\lambda)}}{\mu_2\left[\mu_2-2\lambda+\sqrt{\mu_2(\mu_2+4\lambda)}\right]} \tag{9}$$

For $N = 3$, (7) is rewritten as

$$x^4 - 3(1+\theta)x^3 + 3(1+\theta)^2x^2 - (1+\theta)^3x + \theta^3 = 0$$

By extracting the factor $(x-1)$ of the above equation, we have

$$(x-1)\left[x^3 - (2+3\theta)x^2 + \left(1+3\theta+3\theta^2\right)x - \theta^3\right] = 0$$

which solves to yield

$$\begin{aligned} x &= \tfrac{(2+3\theta)}{3} + \sqrt[3]{V + \sqrt[2]{V^2+W^3}} + \sqrt[3]{V - \sqrt[2]{V^2+W^3}} \\ &= \left(\tfrac{2}{3} + \tfrac{\lambda}{\mu_3}\right) + \sqrt[3]{V + \sqrt[2]{V^2+W^3}} + \sqrt[3]{V - \sqrt[2]{V^2+W^3}} \end{aligned} \tag{10}$$

in which

$$\begin{aligned} V &= \tfrac{-(2+3\theta)\left(1+3\theta+3\theta^2\right)}{6} + \tfrac{(2+3\theta)^3}{27} + \tfrac{\theta^3}{2} \\ &= \tfrac{-\left(2+\frac{9\lambda}{\mu_3}+27\left(\frac{\lambda}{\mu_3}\right)^2\right)}{54} \\ &= \tfrac{2{\mu_3}^2+9\lambda\mu_3+27\lambda^2}{54{\mu_3}^2} \end{aligned}$$

and

$$\begin{aligned} W &= \tfrac{\left(1+3\theta+3\theta^2\right)}{3} - \tfrac{(2+3\theta)^2}{9} \\ &= -\tfrac{3\theta+1}{9} \\ &= -\tfrac{3\lambda+\mu_3}{9\mu_3} \end{aligned}$$

Note that from the Galois's theorem, (7) does not have a close form for $N > 3$ and must be computed numerically. As we pointed out previously, the second P4 switch can accommodate traffic ρ_N if we provide a small buffer space β to satisfy the requirement $E[t_W] = \beta/\mu_N$. In other words, if the second P4 switch can accommodate ρ_N, its serving rate μ_N (computing power) must satisfy

$$E[t_W] \leq \frac{\beta}{\mu_N} \tag{11}$$

Suppose that the original design of the second P4 switch targets at the line rate μ_1, in which the packets are not aggregated, and every packet is processed at a fixed service time $1/\mu_1$. We also consider exponential service time with the same rate $1/\mu_1$ for the comparison and validation purpose. For an arbitrary packet arrival rate λ, if we want to keep the line rate μ_1 to the switch without packet aggregation/disaggregation, then the inequality (11) must be satisfied, that is, from the response time of an M/M/1 queue, we have

$$\frac{\lambda}{\mu_1(\mu_1-\lambda)} \leq \frac{\beta}{\mu_1} \text{ or } \mu_1 \geq \left(\frac{\beta+1}{\beta}\right)\lambda \tag{12}$$

If the packet processing time is a fixed value, then from the response time of an M/D/1 queue, we have

$$\frac{\lambda}{2\mu_1(\mu_1-\lambda)} \leq \frac{\beta}{\mu_1} \text{ or } \mu_1 \geq \left(\frac{2\beta+1}{2\beta}\right)\lambda \tag{13}$$

Let $\mu_1^*(\beta)$ be the minimal μ_1 that satisfies (12) and (13). Then we have

$$\mu_1^*(\beta) = \begin{cases} \left(\frac{\beta+1}{\beta}\right)\lambda & \text{for Exponential service times} \\ \left(\frac{2\beta+1}{2\beta}\right)\lambda & \text{for fixed service times} \end{cases} \tag{14}$$

Equation (14) indicates that the system with the fixed service times can process packets with less computing power than that for exponential service times. We have developed event-driven simulation to model the second P4 switch's behavior. The event driven simulation is the same as those for the G/D/1 and the G/M/1 queues built in [18–21], and the details are omitted. Equation (14) is used to validate the simulation model. Our experiments indicate that the discrepancies between the analytic and the simulation results are within 0.4%.

Now, we consider the case in which the packets are aggregated into 2-packets at the first P4 switch, and are sent to the second P4 switch for disaggregation. That is, for $N = 2$, substitute (9) into (11) to yield

$$\frac{\mu_2 + 2\lambda - \sqrt{\mu_2(\mu_2 + 4\lambda)}}{\mu_2\left[\mu_2 - 2\lambda + \sqrt{\mu_2(\mu_2 + 4\lambda)}\right]} \leq \frac{\beta}{\mu_2}$$

which is simplified as

$$\left[\lambda - \frac{\beta\mu_2}{1+\beta} - \left(\sqrt{\beta^2+\beta}\right)\left(\frac{\mu_2}{1+\beta}\right)\right]\left[\lambda - \frac{\beta\mu_2}{1+\beta} + \left(\sqrt{\beta^2+\beta}\right)\left(\frac{\mu_2}{1+\beta}\right)\right] \leq 0$$

The above inequality implies

$$\lambda - \frac{\beta\mu_2}{1+\beta} - \left(\sqrt{\beta^2+\beta}\right)\left(\frac{\mu_2}{1+\beta}\right) \leq 0$$

which is solved to yield

$$\lambda \leq \left(\frac{\beta + \sqrt{\beta^2+\beta}}{1+\beta}\right)\mu_2$$

Let $\mu_2^*(\beta)$ be the minimal μ_2 that satisfies the above inequality. Then,

$$\mu_2^*(\beta) = \frac{(1+\beta)\lambda}{\beta + \sqrt{\beta^2+\beta}} \tag{15}$$

Similarly, for $N = 3$, substitute (10) into (11) to yield

$$\frac{(2\mu_3 + 3\lambda) + 3\mu_3\left(\sqrt[3]{V + \sqrt[2]{V^2+W^3}} + \sqrt[3]{V - \sqrt[2]{V^2+W^3}}\right)}{\mu_3\left[\mu_3 - 3\lambda - 3\mu_3\left(\sqrt[3]{V + \sqrt[2]{V^2+W^3}} + \sqrt[3]{V - \sqrt[2]{V^2+W^3}}\right)\right]} \leq \frac{\beta}{\mu_3}$$

Let $\mu_3^*(\beta)$ be the minimal μ_3 that satisfies the above inequality. Then,

$$\mu_3^*(\beta) = \frac{3(1+\beta)\lambda}{(\beta-2) - 3(1+\beta)\left(\sqrt[3]{V + \sqrt[2]{V^2+W^3}} + \sqrt[3]{V - \sqrt[2]{V^2+W^3}}\right)} \tag{16}$$

Equations (15) and (16) are used to validate the simulation model again. Our experiments indicate that the discrepancies between the analytic and the simulation results are within 0.9%.

After validation by (15) and (16), the simulation model, together with the measured $\mu_N^*(\beta)$ obtained from Figure 9 and Equation (5), are used to compute $\mu_N^*(\beta)$, the minimal μ_N that satisfies (11). Figure 10 plots $\mu_N^*(\beta)$ against the aggregation factor N and the queueing factor β, in which $\mu_N^*(\beta)$ is normalized by $\mu_N^*(0)$. The figure indicates that a queueing mechanism with a small buffer can significantly reduce the computation power required to handle disaggregation of the IoT packets. Specifically, when $\beta \leq 0.02$, increasing β will significantly reduce $\mu_N^*(\beta)$. On the other hand, for $\beta \geq 0.05$, increasing the buffer size does not improve the performance. Therefore, a buffer of size $\lceil \beta \rceil = 1$ is good enough to reduce the processing rate to $\mu_N^*(\beta)$, which is not difficult to implement in the P4 chip (i.e., to create a buffer at Figure 4(9)).

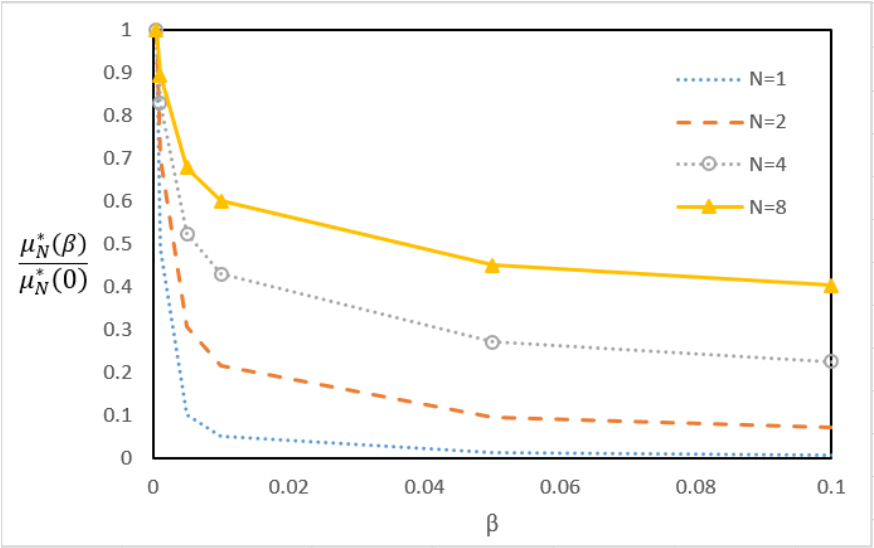

Figure 10. $\mu_N^*(\beta)$ against N and β.

5. Related Work and Aggregation/Disaggregation Based on SDN Controllers

In the existing solutions, packet aggregation/disaggregation is performed by software run at the CPUs of the servers. As we show in this section, the software solutions based on the CPU-based architecture will result in degraded goodput (and therefore degraded throughput). Many studies [10,22,23] described the benefits of packet aggregation in saving transmission overhead in the networks. However, none of them investigate the processing cost of a network node for aggregation/disaggregation. This section shows the aggregation/disaggregation cost in the servers based on the CPU architecture.

The original design of SDN separates the control plane and the data plane of legacy switches so that the network nodes (e.g., SDN switches) are only responsible for data forwarding, which makes the network management simpler and more convenient. The SDN controller called the OpenFlow Controller (OFC) utilizes OpenFlow [24] to interact with the SDN switches called the OpenFlow Switches (OFSs). SDN network management introduces a flow control application that maintains the routing rules of the OFSs through the OFC. In this way, SDN makes the network more programmable, flexible, and dynamically manageable. Specifically, the SDN can dynamically adjust the data routing path to avoid network congestion. Figure 11 shows an aggregation/disaggregation mechanism for IoTtalk based on the SDN architecture, which is similar to the one illustrated in Figure 5, except that the P4 switches are replaced by the OpenFlow switches (which cannot be programmed to perform packet aggregation/disaggregation), and the OpenFlow Controllers are involved. In this architecture, K IoT devices (Figure 11(1)) are installed in Building 311 of the MIRC Building. The DAs of the devices collect the sensor data and send small packets (the IoT messages) to an OpenFlow Switch (OFS1; Figure 11(2)). OFS1 connects to an OpenFlow Controller (OFC1; Figure 11(3)) that is responsible for packet aggregation. OFC1 sends the aggregated packets to OFS1. These aggregated packets are sent to another OpenFlow Switch (OFS2; Figure 11(4)) through one or more SDN networks. OFS2 sends the packets to its controller (OFC2; Figure 11(5)) for packet disaggregation and outputs them to the IoTtalk server (Figure 11(6)). Both OFC1 and OFC2 are servers responsible for processing the packets.

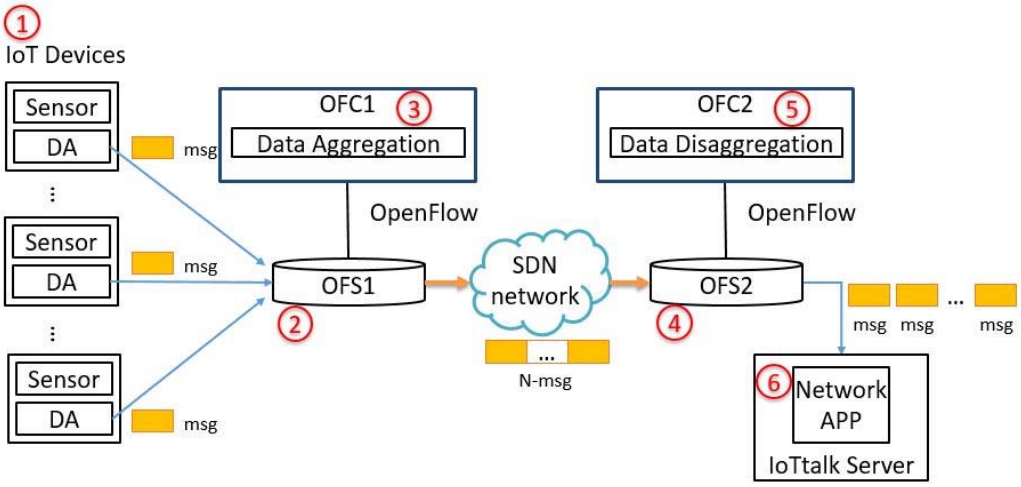

Figure 11. Sensor data aggregation/disaggregation for IoTtalk based on the SDN controllers (the CPU-based architecture).

Figure 12 shows the goodput of aggregation/disaggregation processed by Figure 11(3),(5). The goodput curves indicate that in the CPU-based architecture (i.e., the OFC servers), aggregation/disaggregation processing overheads are significantly higher than those of the pipeline P4 architecture shown (Figure 9). Specifically, the goodputs for $N \geq 2$ are worse than those for $N = 1$ in Figure 12. On the other hand, the goodputs for $N \geq 2$ are much better than those for $N = 1$ in Figure 10.

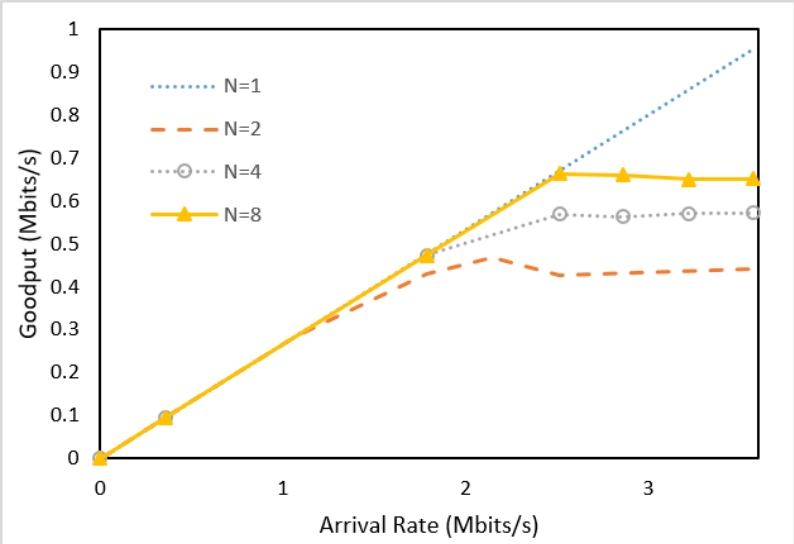

Figure 12. Goodput vs. packet arrival rates for the SDN controller configuration ($1 \leq N \leq 8$).

6. Concluding Remarks

This paper proposed to utilize the P4 switch for quick packet aggregation and disaggregation. We designed and developed novel P4 programs for aggregation and disaggregation, which are implemented in commercial P4 switches. Our study indicated that packet aggregation can be achieved in a P4 switch with its 100 Gbps line rate (without extra packet processing cost). On the other hand, to disaggregate a packet that combines N IoT messages, the processing time is about the same as processing N individual IoT messages. We further proposed to provide a small buffer at the input ports of the P4 switch to significantly reduce the packet drop probability when disaggregating an N-packet. Also note that our implementation conducted IoT message aggregation at the highest bit rate (100 Gbps) that has not been found in the literature. As for the future work, we are considering

new techniques to improve the performance of packet disaggregation. One possible technique is the multicast mechanism supported by the EdgeCore switch.

As a final remark, if the aggregation and the disaggregation P4 switches are indirectly connected through a network with bandwidth lower than their line rates, due to network congestion, when packets travel from the first P4 switch to the second P4 switch, some packets may be dropped while the aggregated N-packets are safely delivered to the second switch. In this case, together with the disaggregation process performed at the second P4 switch, the aggregation process performed at the first P4 switch effectively reduces the probability of IoT messages lost over the lower-bandwidth network.

Author Contributions: Y.-B.L. is responsible for Analytic model; S.-Y.W. is responsible for Design and Data curation; C.-C.H. is responsible for Simulation and Data analysis; C.-M.W. is responsible for P4 switch programming.

Funding: This research was funded by Ministry of Education: Center for Open Intelligent Connectivity from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lin, Y.-B.; Lin, Y.-W.; Chih, C.-Y.; Li, T.-Y.; Tai, C.-C.; Wang, Y.-C.; Lin, F.J.; Kuo, H.-C.; Huang, C.-C.; Hsu, S.-C. EasyConnect: A Management System for IoT Devices and Its Applications for Interactive Design and Art. *IEEE Internet Things J.* **2015**, *2*, 551–561. [CrossRef]
2. Lin, Y.-B.; Lin, Y.-W.; Huan, C.-M.; Chih, C.-Y.; Lin, P. IoTtalk: A Management Platform for Reconfigurable Sensor Devices. *IEEE Internet Things J.* **2017**, *4*, 1552–1562. [CrossRef]
3. Lin, Y.-W.; Lin, Y.-B.; Yang, M.-T.; Lin, J.-H. ArduTalk: An Arduino Network Application Development Platform based on IoTtalk. *IEEE Syst. J.* **2017**, *PP*, 1–9. [CrossRef]
4. Lin, Y.-W.; Lin, Y.-B.; Hsiao, C.-Y.; Wang, Y.-Y. IoTtalk-RC: Sensors as Universal Remote Control for Aftermarket Home Appliances. *IEEE Internet Things J.* **2017**, *4*, 1104–1112. [CrossRef]
5. Open Connectivity Foundation. Available online: https://openconnectivity.org/ (accessed on 22 May 2018).
6. oneM2M. Standards for M2M and the Internet of Things. Available online: http://www.onem2m.org/ (accessed on 22 May 2018).
7. TS 102 921—V1.3.1—Machine-to-Machine Communications (M2M); mIa, dIa and mId Interfaces. Available online: http://www.etsi.org/deliver/etsi_ts/102900_102999/102921/01.03.01_60/ts_102921v010301p.pdf (accessed on 22 May 2018).
8. Wang, S.-Y.; Chen, Y.-R.; Chen, T.-Y.; Chang, C.-H.; Cheng, Y.-H.; Hsu, C.-C.; Lin, Y.-B. Performance of LoRa-based IoT Applications on Campus. In Proceedings of the IEEE Vehicular Technology Conference, Toronto, ON, Canada, 24–27 September 2017.
9. Lin, Y.-B.; Lin, Y.-W.; Hsiao, C.-Y.; Wang, S.-Y. Location-based IoT applications on campus: The IoTtalk approach. *Pervasive Mob. Comput.* **2017**, *40*, 660–673. [CrossRef]
10. Koike, A.; Ohba, T.; Ishibashi, R. IoT Network Architecture Using Packet Aggregation and Disaggregation. In Proceedings of the IIAI International Congress on Advanced Applied Informatics, Kumamoto, Japan, 10–14 July 2016.
11. ONF SDN Evolution. Available online: http://opennetworking.wpengine.com/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf (accessed on 7 June 2018).
12. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
13. P4 Language Spec Version 1.0.4. Available online: https://p4lang.github.io/p4-spec/p4-14/v1.0.4/tex/p4.pdf (accessed on 22 May 2018).
14. P4. Available online: http://www.p4.org/ (accessed on 22 May 2018).
15. Wu, C.-M.; Wang, S.-Y.; Lin, Y.-B.; Huang, C.-C. Design and Implementation of Packet Aggregation by P4 Switches. In Proceedings of the IEEE Global Communications Conference, Abu Dhabi, UAE, 9–13 December 2018. under review.

16. Nelson, R. *Probability, Stochastic Process, and Queueing Theory*; Springer: New York, NY, USA, 1995; ISBN 978-0-387-94452-4.
17. Spirent TestCenter—Verifying Network and Cloud Evolution. Available online: https://www.spirent.com/Products/TestCenter (accessed on 22 May 2018).
18. Huang, D.-W.; Lin, P.; Gan, C.-H. Design and performance study for a mobility management mechanism (WMM) using location cache for wireless mesh network. *IEEE Trans. Mob. Comput.* **2008**, *7*, 546–556. [CrossRef]
19. Hong, C.-Y.; Pang, A.-C. 3-approximation algorithm for joint routing and link scheduling in wireless relay networks. *IEEE Trans. Wirel. Commun.* **2009**, *8*, 856–861. [CrossRef]
20. Lin, P.; Wu, S.-H.; Chen, C.-M.; Liang, C.-F. Implementation and performance evaluation for a ubiquitous and unified multimedia messaging platform. *ACM Wirel. Netw.* **2009**, *15*, 163–176. [CrossRef]
21. Tsai, S.-Y.; Sou, S.-I.; Tsai, M.-H. Reducing energy consumption by data aggregation in M2M networks. *Wirel. Pers. Commun.* **2014**, *74*, 1231–1244. [CrossRef]
22. Yousefi, H.; Yeganeh, M.-H.; Alinaghipour, N.; Movaghar, A. Structure-free real-time data aggregation in wireless sensor networks. *Comput. Commun.* **2012**, *35*, 1132–1140. [CrossRef]
23. Fan, K.-W.; Liu, S.; Sinha, P. Structure-Free Data Aggregation in Sensor Networks. *IEEE Trans. Mob. Comput.* **2007**, *6*, 929–942. [CrossRef]
24. OpenFlow® Switch Specification Ver 1.3.5. Available online: http://opennetworking.wpengine.com/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf (accessed on 7 June 2018).