УДК 654.027.3

Модель генерации и обработки трафика IoT параллельными коммутационными системами

Д. В. Кутузов[1], А. В. Осовский, О. В. Стукач

В статье описываются особенности генерации и обработки трафика IoT параллельными пространственными коммутационными системами. Концепция технологии интернета вещей (IoT) предполагает совмещение функций маршрутизации с функциями обработки трафика. Такое совмещение хорошо прослеживается на технологии «сеть на кристалле» (NoC). Ядром большинства систем NoC являются параллельные пространственные коммутационные системы с матрицами коммутации 5×5. Упрощенная модель коммутатора NoC была нами реализована в виде системы массового обслуживания. Для реализации модели использовался язык программирования Python. В исследовании нами были изучены особенности обработки трафика IoT параллельной пространственной коммутационной системой, имеющей бесконечно большие входные и выходные буферы. В результате работы были получены значения максимальных и средних очередей в выходных буферных устройствах системы.

Ключевые слова: интернет вещей, параллельная коммутация, сеть на кристалле.

1. Введение

Технология интернета вещей (Internet of Things, IoT) постепенно внедряется во многие сферы человеческой деятельности. По различным прогнозам в ближайшее время ожидается включение в инфокоммуникационную структуру порядка 30 млрд умных объектов (вещей), являющихся составными единицами технологии IoT. Широкое распространение уже получили такие приложения технологии IoT, как умные дома и мониторинг здоровья [1]. Бурно развивается и программное обеспечение, позволяющее полнее использовать возможности технологии IoT, стимулирующее рынок и расширяющее сферы применения технологии.

Вместе с тем включение в информационные сети умных объектов выявило ряд особенностей обработки информационного трафика от них. Ниже мы рассматриваем наиболее распространенные модели обработки трафика IoT, особенности построения средств коммутации и предлагаем имитационную модель, позволяющую исследовать характеристики обработки трафика IoT пакетными коммутаторами.

2. Обсуждение

Исследованию особенностей генерации и регулирования трафика IoT посвящено большое количество статей. Так в работе [2] авторы утверждают, что появление большого количества умных устройств приводит к возникновению мгновенно нарастающего числа коротких сеансов. Это приводит к необходимости создания формальных моделей, гибко отражающих особенности возникающего в реальных сетях IoT-трафика.

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00612.

Пример построения одной из моделей трафика IoT приведен в [3]. В статье рассмотрен поток заявок на обслуживание, возникающий на ограниченном временном интервале. По мнению авторов, моделями, наилучшим образом представляющими поток заявок в пакетных сетях передачи данных, являются так называемые модели, «не имеющие хвостов». В выводах своего исследования, авторы приводят средние значения времени задержки заявок и коэффициент вариации этой же случайной величины, которые превышают аналогичные величины, свойственные моделям с «тяжелыми хвостами».

В статье [4] авторы предлагают модель гибридного планирования приоритетов заявок для служб с различными функциями и ограничениями QoS. Согласно этой стратегии сервисам, чувствительным к задержкам, назначается высокий приоритет, и они выполняются немедленно, а сервисы, нечувствительные к задержкам, предоставляются без приоритета. Анализ такой модели обслуживания заявок показал, что её использование приводит к увеличению длины очереди ожидания обслуживания.

Одна из моделей трафика устройств IoT изложена в статье [5], где описывается применение беспроводных сенсорных сетей в медицине для контроля ЭКГ и температуры тела пациентов. Для описания модели трафика подобных устройств авторы использовали два процесса, один из них с максимальной скоростью – ЭКГ, второй – с минимальной скоростью – изменение температуры тела. На основе анализа трафика авторы пришли к выводу, что модель источника трафика имеет гауссовское распределение.

Для обработки возрастающего трафика IoT требуются технологии построения современных средств коммутации и маршрутизации, позволяющие его обрабатывать с высоким качеством. Например, в работах [6–8] описываются технологические подходы к созданию материалов и элементов, которые могут использоваться в перспективных коммутаторах трафика IoT. Ряд исследований направлен на эффективное использование буферных устройств коммутаторов для повышения пропускной способности коммутаторов [9]. Также некоторые авторы исследуют возможности и формулируют принципы построения перспективных оптических коммутаторов [10–12].

Архитектурным решением, позволяющим не только выполнять функции коммутации, но и производить обработку данных непосредственно на кристалле, является технология NoC [13–15]. В статье [16] мы показали, что подобные системы могут использоваться как основа сетей IoT, где распределение трафика между узлами совмещается с его обработкой в этих узлах, что соответствует концепции технологий IoT и 5G. Мы также предложили архитектуру параллельных коммутационных систем [17], которые могут использоваться в составе маршрутизаторов IoT, алгоритмы функционирования таких систем [18] и провели анализ их функционирования на основе классической [19] и таймированной (timed) сети Петри CPN [20].

3. Имитационная модель

Для представления системы массового обслуживания (СМО) в виде программной модели мы использовали язык программирования Python. К достоинствам этого языка можно отнести кроссплатформенность, а также наличие большого количества свободно распространяемых библиотек.

Для имитационного моделирования СМО одной из таких библиотек является queueingtool. Эта библиотека представляет собой простой агентный симулятор систем массового обслуживания. Моделирование основано на событиях, где события состоят в числе прибытий и убытий агентов (заявок, информационных пакетов, транзактов), которые переходят из очереди в очередь в сети СМО. Однако только этой библиотеки недостаточно для создания модели СМО. Поэтому мы использовали еще некоторые другие библиотеки, такие как NetworkX, NumPy и Matplotlib. Библиотека NetworkX необходима для представления сети СМО в виде графа и выполнения действий, связанных с атрибутами этого графа, например, такими как

использование матрицы весов, визуализации графа и т.д. Библиотека NumPy также необходима для работы queueuing-tool, выполняя обработку и преобразование данных. В случае если нам требуется визуализация полученных результатов в виде графиков, нам необходима библиотека Matplotlib.

Для исследования особенностей обработки трафика IoT мы использовали коммутатор с матрицей 5×5, полагая, что обработка трафика ведется коммуникационными средствами технологии NoC или подобной ей (рис. 1). Каждый маршрутизатор NoC имеет по пять входных и пять выходных портов, условно соответствующих направлениям на север, восток, юг и запад, а также локальный элемент обработки (processing element, PE). Работа маршрутизатора состоит в том, чтобы направлять приходящие пакеты с каждого входного порта на необходимый выходной порт, и так шаг за шагом, к конечным пунктам назначения. Для реализации этой функции маршрутизатор оснащен входным буфером для каждого входного порта, коммутатором 5×5 для перенаправления трафика на желаемый выходной порт и логикой управления для обеспечения функций маршрутизации.

На рис. 2 представлена модель СМО, представляющая такой коммутатор.

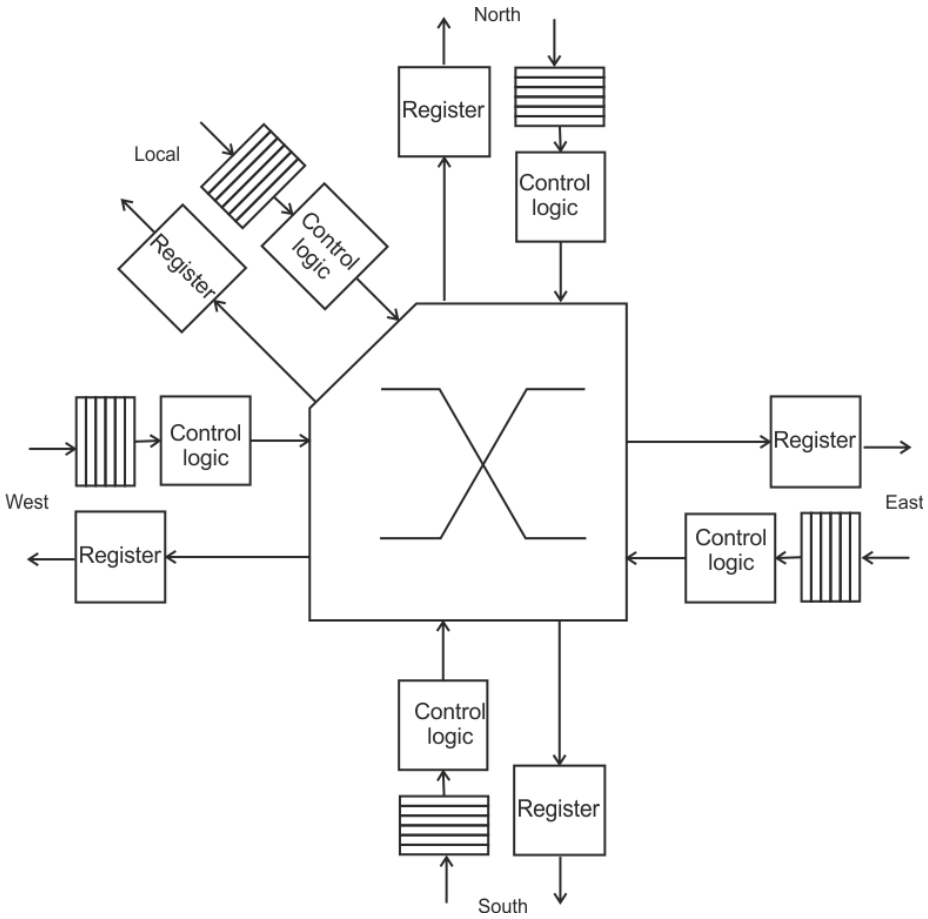

Рис. 1. Коммутатор сети обработки данных технологии NoC

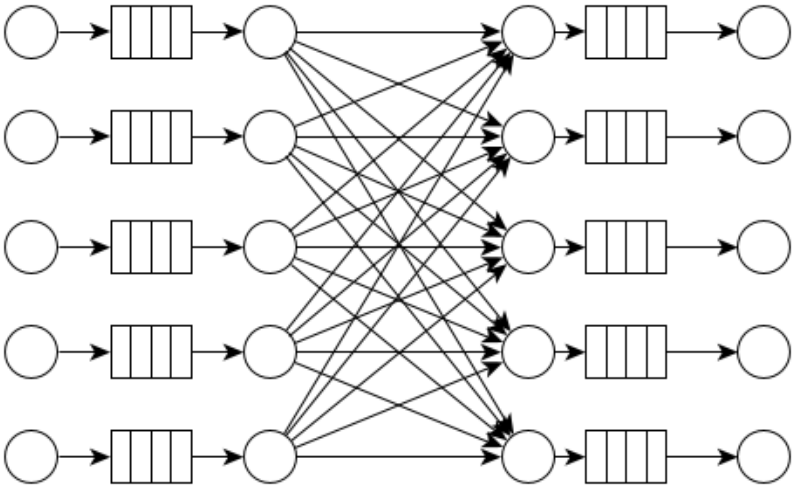

Рис. 2. Модель сети массового обслуживания, представляющая коммутатор технологии NoC

Особенностью библиотеки queueuing-tool является представление очередей ребрами графа, поэтому традиционное графическое представление систем массового обслуживания (рис. 2), было нами преобразовано в граф, представленный на рис. 3.

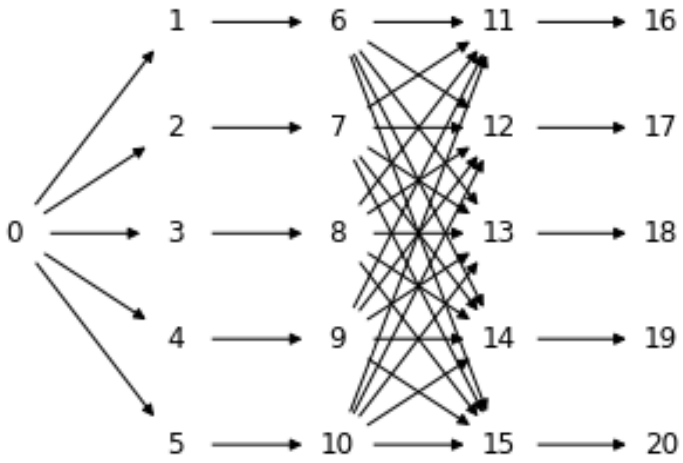

Рис. 3. Сеть СМО, использующаяся в имитационной модели

Для задания сети, представленной на рисунке, использовался следующий программный код:

```
ad = {0: [1, 2, 3, 4, 5],
    1: [6],
    2: [7],
    3: [8],
    4: [9],
    5: [10],
    6: [11, 12, 13, 14, 15],
    7: [11, 12, 13, 14, 15],
    8: [11, 12, 13, 14, 15],
    9: [11, 12, 13, 14, 15],
    10: [11, 12, 13, 14, 15],
    11: [16],
```

```
 12: [17],
 13: [18],
 14: [19],
 15: [20]
}
```

Сеть обслуживания определяется структурой ad, которая представляет собой словарь, где ключи – номера вершин-источников, а значения – списки номеров вершин-приемников.

Как уже сообщалось выше, помимо структуры сети нам необходимо было также задать типы ребер графа, что выполняется следующим образом:

```
ade = {0: {1: 1, 2: 1, 3: 1, 4: 1},
 1: {5: 2},
 2: {6: 2},
 3: {7: 2},
 4: {8: 2},
 5: {9: 3, 10: 3, 11: 3, 12: 3},
 6: {9: 3, 10: 3, 11: 3, 12: 3},
 7: {9: 3, 10: 3, 11: 3, 12: 3},
 8: {9: 3, 10: 3, 11: 3, 12: 3},
 9: {13: 4},
 10: {14: 4},
 11: {15: 4},
 12: {16: 4}
}
```

Переменная ade представляет собой словарь словарей, где ключи основного словаря, как и в предыдущей переменной ad, – это номера вершин-источников, ключи вложенных словарей – номера вершин-приемников, а значения вложенных словарей – тип ребра, соединяющего источник и приемник. Здесь используются ребра с типами 1, 2, 3 и 4. Кроме того, неявно используются ребра типа 0. Согласно описанию библиотеки используются следующие обозначения типов ребер: тип 1 – может принимать агенты извне сети, фактически среда, откуда поступают заявки; тип 2, 3, 4 и т.д. – очереди, свойства которых необходимо будет определить, тип 0 – терминальные ребра, его можно не задавать, они будут созданы автоматически как петли в вершинах 16 – 20.

Далее заданные нами структуры необходимо преобразовать в структуры, с которыми сможет работать библиотека:

```
g = qt.adjacency2graph(adjacency=ad, edge_type=ade)
ans = qt.graph2dict(g)
```

Функция qt.adjacency2graph преобразует заданные нами структуры в граф, в котором у каждого ребра/очереди есть тип, причем ребра (0, 1), (0, 2), (0, 3), (0, 4) и (0, 5) являются источниками случайных потоков заявок, поступающих извне сети, в соответствии с необходимым нам законом распределения (мы зададим его позже). Остальные ребра представляют собой очереди с теми или иными характеристиками, такими как время задержки заявки, количество устройств обслуживания и др., которые мы также зададим позже.

Функция qt.graph2dict – преобразует граф в словарь, формат которого использует библиотека.

Для графического представления сети обслуживания (рис. 3) можно воспользоваться функцией draw_networkx библиотеки NetworkX, но вначале необходимо задать позиции вершин графа в сетке координат, иначе в сети трудно будет ориентироваться. Определим позиции вершин для рисования графа:

```
pos = {0: (-0.25, 0),
  1: (1, 0.75), 2: (1, 0.25), 3: (1, -0.25), 4: (1, -0.75), 5: (1, -1.25),
```

```
  6: (2, 0.75), 7: (2, 0.25), 8: (2, -0.25), 9: (2, -0.75), 10: (2, -1.25),
  11: (3, 0.75), 12: (3, 0.25), 13: (3, -0.25), 14: (3, -0.75), 15: (3, -1.25),
  16: (4, 0.75), 17: (4, 0.25), 18: (4, -0.25), 19: (4, -0.75), 20: (4, -1.25)
   }
```

Прорисовываем граф и записываем его изображение в файл stuct.png:

```
nx.draw_networkx(g, pos, with_labels=True, node_size=900, node_color='w')
plt.savefig("stuct.png")
```

После того как определена структура сети обслуживания, необходимо определить параметры сети – то, как генерируются заявки (здесь arr_f(t)), и время обработки заявки в системе (здесь ser_f(t)). Время обработки заявки также может быть задано как случайный процесс с определенным законом распределения. Обе функции определяются как функции времени:

```
def arr_f(t):
  return t + np.random.normal(loc=my_lambda, scale= my_sigma)

def ser_f(t):
  return t + my_time
```

Теперь необходимо определить типы ребер и характеристики обслуживания:

```
q_classes = {1: qt.QueueServer, 2: qt.QueueServer, 3: qt.QueueServer, 4: qt.QueueServer}

q_args = {1:
      {'arrival_f': arr_f,
       'service_f': lambda t: t,
       'AgentFactory': qt.GreedyAgent
        },
      2:
      {'num_servers': 1,
       'service_f': lambda t: t
      },
      3:
      {'num_servers': 1,
      'service_f': lambda t: t
      },
      4:
      {'num_servers': 1,
       'service_f': ser_f
      },
  }
```

Все ребра представляют собой структуры типа QueueServer, а характеристики определяются типом ребер (1, 2, 3 или 4).

Создаем СМО:

```
qn = qt.QueueNetwork(g=g, q_classes=q_classes, q_args=q_args, seed=13)
qn.g.new_vertex_property('pos')
qn.g.set_pos(pos)
```

Для созданной системы массового обслуживания вызываем следующие функции:

```
qn.start_collecting_data()
```

– инициируем сбор данных от агентов сети;

```
qn.initialize(edge_type=1)
```

– инициализируем сеть, сообщая, что агенты будут поступать на ребра типа 1;

```
qn.simulate(t=time_simulate)
```

– задаем время симуляции.

Производим сбор данных от агентов:

```
qn.num_events
results = qn.get_queue_data(edge=[(vertex_source, vertex_receiver)])
```

Данные от агентов, собранные на ребре edge с началом в вершине vertex_source и концом в вершине vertex_receiver, будут помещены в переменную results.

Функция get_queue_data возвращает данные в виде словаря, где ключи являются идентификатором агента agent_id, а значения – массивом данных этого агента. Столбцы этого массива выглядят следующим образом: первый – время прибытия агента; второй – время начала работы (обслуживания) агента; третий – время выбытия агента; четвертый – длина очереди при поступлении агентов; пятый – общее количество агентов в QueueServer; шестой - идентификатор QueueServer.

После прогона модели собранные данные и СМО необходимо очищать:

```
qn.clear_data()
qn.clear()
```

4. Результаты моделирования

Для исследования особенностей обработки трафика IoT параллельными коммутационными системами мы использовали агентную модель СМО, в которой генерация агентов (фактически – заявок на обслуживание, пакетов) производится в соответствии с гауссовским распределением. Мы использовали такие параметры генерации потока заявок: среднее значение между поступлением агентов в единицах модельного времени (MTU) μ, среднеквадратичное отклонение σ, длительность обслуживания агента τ.

Интенсивность потока заявок в предлагаемой модели значительно выше, чем, например, в модели [5]. Это объясняется необходимостью изучить особенности обработки трафика в критических режимах работы маршрутизатора IoT и его ядра – параллельного матричного коммутатора.

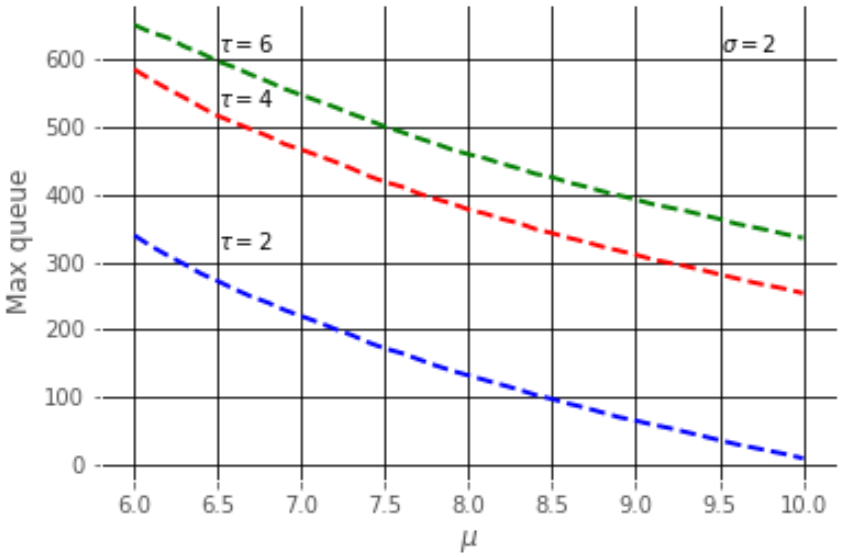

Рис. 4. Зависимость максимальной очереди от интенсивности потока заявок

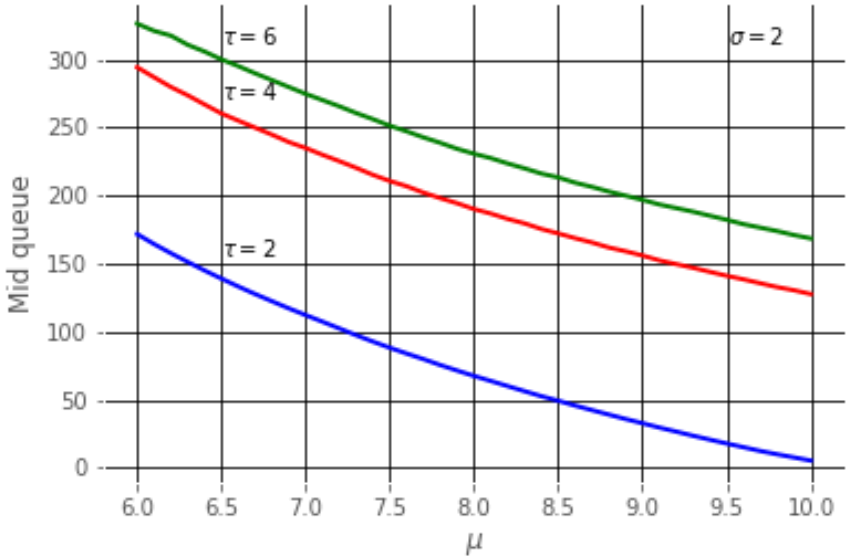

Рис. 5. Зависимость средней очереди от интенсивности потока заявок

Мы провели эксперименты и изучили максимальные и средние очереди в выходном буфере коммутатора (на рис. 3 он представлен ребром (11, 16)), изменяя интенсивность поступающего трафика. Время обслуживания каждого агента мы выбрали постоянным, что характерно для трафика, поступающего от датчиков IoT. На рисунках представлены зависимости длин очередей от среднего времени между поступлением заявок на обслуживание μ для различных значений времени обслуживания τ (при σ = 2).

5. Заключение

В этом исследовании мы показали особенности обработки потока пакетов от устройств IoT параллельными коммутационными системами. Рассмотренная здесь модель типичного коммутатора технологии NoC нами была значительно упрощена, так как реальные устройства имеют весьма сложную структуру, учесть особенности которой крайне сложно в рамках имитационного моделирования. Для генерации заявок (пакетов) мы использовали гауссовскую модель распределения вероятностей, которая является наиболее распространенной для изучения трафика IoT. Предложенная нами модель при незначительных изменениях, таких как применение других законов распределения помимо гауссовского, применении более сложных алгоритмов обслуживания, задержек в сети, может использоваться и для изучения других режимов работы подобных устройств.

Литература

1. *Perry Lea.* Internet of Things for architects: architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security. Publicerad: 2018, Birmingham: Packt Publishing Ltd. Copyright: 2018. 505 p. ISBN 9781788470599.
2. *Wei-Hung Hsu, Qiuhui Li, Xue-Hai Han, and Chih-Wei Huang.* A Hybrid IoT Traffic Generator for Mobile Network Performance Assessment // 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC).
3. *Levakov A. K., Sokolov A. N., Sokolov N. A.* Models of incoming traffic in packet networks // T-Comm. 2015. V. 9, № 5. P. 91–94.

4. *Wen-Xiang Li; Jun Xu; Hao Jiang.* Queuing States Analysis on a Hybrid Scheduling Strategies for Heterogeneous Traffics in IOT // 2012 International Conference on Computer Science and Service System.
5. *Messier G G., Finvers I. G.* Traffic Models for Medical Wireless Sensor Networks // IEEE Communications Letters. 2007. V. 11, № 1. P. 13–15. DOI: 10.1109/LCOMM.2007.061291, URL: https://ieeexplore.ieee.org/document/4114210
6. *Starov D., Kutuzov D., Stukach O., Osovskiy A.* Measuring complex for studying galvanomagnetic phenomena in multigrafene layers // 2017 International Siberian Conference on Control and Communications (SIBCON), DOI: 10.1109/SIBCON.2017.7998529, URL: http://ieeexplore.ieee.org/document/7998529/
7. *Vytovtov K., Barabanova E., Zouhdi S.* Penetration effect in uniaxial anisotropic metamaterials // Appl. Phys. A. 2018. P. 124–137. https://doi.org/10.1007/s00339-018-1563-z
8. *Vytovtov K., Barabanova E.* Unusual penetration effect in ferromagnetics. Negative refraction under tangential wave incidence // Journal of Physics: Conf. Series. 2018. 1092 (1), 012164. DOI :10.1088/1742-6596/1092/1/012164
9. *Georgakopoulos G. F.* Buffered Crossbar Switches, Revisited: Design Steps, Proofs and Simulations Towards Optimal Rate and Minimum Buffer Memory // IEEE ACM Transactions on networking. 2008. V. 16, № 6. DOI: 10.1109/TNET.2007.911441. URL: http://ieeexplore.ieee.org/document/4460578/
10. *Vytovtov K. A., Barabanova E. A., and Podlazov V. S.* Model of Next-Generation Optical Switching System Distributed Computer and Communication Networks // 21st International Conference DCCN 2018, Moscow, Russia, September 17–21, 2018. P. 377–386.
11. *Barabanov I. O., Maltseva N. S., Barabanova E. A.* Switching cell for information transmission optical systems // 2016 International Conference on Actual Problems of Electron Devices Engineering (APEDE 2016). P. 343–347. DOI: 10.1109/APEDE.2016.7879025, URL: https://ieeexplore.ieee.org/document/7879025
12. *Vytovtov K. A., Barabanova E. A., Barabanov I. O.* Next-generation switching system based on 8x8 self-turning optical cell // International Conference on Actual Problems of Electron Devices Engineering (APEDE 2018). P. 306–310.
13. *Anuja Naik, Tirumale K. Ramesh.* Efficient Network on Chip (NoC) using heterogeneous circuit switched routers // 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), Bangalore, India. 10–12 Jan. 2016. DOI: 10.1109/VLSI-SATA.2016.7593043, URL: https://ieeexplore.ieee.org/document/7593043
14. *Wenjie Li, Yiping Gong, Bin Liu.* Performance Evaluation of Crossbar Switch Fabrics in Core Routers // 17th International Conference on Advanced Information Networking and Applications (AINA), Xi'an, China, 29–29 March 2003. DOI: 10.1109/AINA.2003.1192996, URL: https://ieeexplore.ieee.org/document/1192996
15. *Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen.* Networks on Chips: Structure and Design Methodologies // Journal of Electrical and Computer Engineering. 2012. Article ID 509465. URL: https://doi.org/10.1155/2012/509465
16. *Kutuzov D. V., Osovsky A. V., Starov D. V., Motorina E. A.* Development of parallel switching facilities for 5G communication systems // Radiotekhnika. 2019. V. 83, № 3. 2019. P. 70–79.
17. *Kutuzov D., Utesheva A.* Switching Element for Parallel Spatial Systems // International Siberian Conference on Control and Communications (SIBCON-2011), Krasnoyarsk, September 15−16, 2011. P. 60–62. URL: http://ieeexplore.ieee.org/document/6072595/
18. *Kutuzov D., Stukach O.* Algorithms of Parallel Switching for Multistage Schemes // 2013 International Siberian Conference on Control and Communications (SIBCON), Krasnoyarsk, September 12−13, 2013. URL: http://ieeexplore.ieee.org/document/6693642/
19. *Kutuzov D., Osovskiy A., Stukach O.* Modeling of interconnection process in the parallel spatial switching systems // 2016 International Siberian Conference on Control and Communications (SIBCON), Moscow; 12–14 May 2016. URL: http://ieeexplore.ieee.org/document/7491852/

20. *Kutuzov D., Osovsky A., Stukach O., Starov D.* CPN-based model of parallel matrix switchboard // 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT). Moscow, March 14–16, 2018. URL: https://ieeexplore.ieee.org/document/8337180

Статья поступила в редакцию 01.10.2019.

Кутузов Денис Валерьевич
к.т.н., доцент, доцент кафедры «Связь» Астраханского государственного технического университета, e-mail: d_kutuzov@mail.ru.

Осовский Алексей Викторович
к.т.н., доцент, главный научный сотрудник ООО «Фьюче Инжиниринг Лаб», e-mail: a_osovskiy@mail.ru.

Стукач Олег Владимирович
д.т.н., профессор департамента электронной инженерии Московского института электроники и математики им. А. Н. Тихонова НИУ ВШЭ, e-mail: ostukach@hse.ru.

IoT traffic generation and processing model with parallel switching systems

D. Kutuzov, A. Osovskiy, O. Stukach

The article describes the features of the generation and processing of IoT traffic with parallel spatial switching systems. The concept of the Internet of Things (IoT) technology involves routing functions with traffic processing functions combination. This combination is well traced on "Network on Chip" (NoC) technology. The core of most NoC systems are parallel spatial switching systems with 5×5 switching matrices. A simplified model of the NoC switch was implemented by us as a queuing system. To implement the model, the Python programming language was used. In this research, the features of IoT traffic processing with parallel spatial switching system having infinitely large input and output buffers were studied. As a result of the work, the values of the maximum and average queues in output buffer devices of the system were obtained.

Keywords: IoT, parallel routing, parallel devices, QoS, NoC.