Dynamic Protocol Stacks in Smart Camera Networks

Markus Happe, Yujiao Huang, Ariane Keller
Communication Systems Group, ETH Zurich, Switzerland
Email: {markus.happe, yujiao.huang, ariane.keller}@tik.ee.ethz.ch

***Abstract*—The term *Internet of Things* is often used to talk about the trend of embedding microprocessors in everyday devices and connecting them to the Internet. The Internet of Things poses challenging communication requirements since the participating devices are heterogeneous, resource-constrained and operate in an ever changing environment. To cope with those requirements, academic research projects have proposed novel network architectures, such as the Dynamic Protocol Stack (DPS) architecture.**

In this paper, we use smart camera networks as an example of the Internet of Things and evaluate the DPS architecture in this scenario. Our smart camera nodes are implemented as an FPGA-based system-on-chip architecture that uses the DPS architecture for the network communication. We evaluate our smart camera nodes in two case studies. In the first case study, we demonstrate that our proposed smart camera network can track a single object over the field of view of several camera nodes. In the second case study, we show that an adaptive hardware/software mapping of the network functionality can save about 22% of the FPGA resources as compared to a static mapping. The hardware/software mapping can be adapted at a processing delay of a single video frame.

I. Introduction

Nowadays, we observe a trend that more and more computers are embedded into all sorts of objects, such as clothes, household articles and sports equipment. For configuration, monitoring, or entertainment purposes, those compute nodes usually offer a connection to the Internet. This also gives this trend the name *Internet of Things*. In contrast to the traditional Internet, the Internet of Things operates in a highly dynamic environment with changing communication requirements (e.g., privacy levels) and network conditions (e.g., link quality). Additionally, the devices are usually resource-constrained in terms of available area, compute power, and energy. These constraints require a communication firmware with minimal overhead. However, most devices use the standard Internet architecture as their communication platform. While the Internet architecture provides the required functionality, it was never optimized for resource-efficient network communication. Rather it abstracts the heterogeneity of the communication partners and only provides a limited degree of adaptation towards dynamic mobile networks. We believe that the Internet of Things would benefit from a novel, highly dynamic network architecture that is aware of the heterogeneity of its nodes and that is capable of optimizing its communication protocols over time in a fine-grained manner.

Therefore, we investigate how flexible protocol stacks can be used instead of static protocol stacks such as TCP/IP or UDP/IP. Flexible protocol stacks split the networking functionalities into individual functional blocks, which can be dynamically linked with each other in order to form arbitrary protocol stacks. In previous work [1] we presented such a flexible protocol stack architecture, called *Dynamic Protocol Stack (DPS) architecture* as well as EmbedNet, an FPGA-based execution environment for dynamic protocol stacks. EmbedNet supports (i) the autonomous configuration of protocol stacks, and (ii) the dynamic mapping of network functionality to either hardware or software. This allows resource-constrained devices to achieve reasonable network processing performance for arbitrary communication requirements, even though they do not have a powerful processor nor unlimited space for hardware accelerators.

In this paper we extend our work by employing our DPS architecture in the Internet of Things, namely in *smart camera networks*. Smart cameras combine several tasks such as video sensing, processing and communication on a single embedded device [2]. A typical task of a smart camera network is to track an object over the field of view of several camera nodes. In such a scenario they often need to process large amounts of data and communicate with each other at strict real-time constraints. Furthermore, smart camera nodes are usually implemented on embedded devices with limited resources, where the computational expensive tasks are mapped to hardware and the control tasks to software [3]. A flexible network architecture that can adapt itself to the current network condition or communication scenario is highly desirable to meet all communication requirements.

Specifically, this work provides the following contributions:

- A novel smart camera node architecture that is entirely programmed on an FPGA platform and uses the EmbedNet platform for communication.
- A first case study where multiple smart camera nodes autonomously track a single moving object.
- A second case study where multiple objects are tracked in the smart camera network where all network communication is encrypted. Each smart camera node can either track an object locally, using its own camera view, or remotely, using the camera view of another camera node.
- An evaluation of the communication throughput between smart camera nodes and a quantification of the costs for updating the hardware/software mapping of the EmbedNet architecture.

The rest of this paper is structured as follows: Section II presents related work. Section III introduces the general concepts of dynamic protocol stacks and the corresponding adaptive hardware/software mapping. Section IV presents our reconfigurable smart camera node architecture. Then, Section V describes a case study where one object is tracked by several smart camera nodes, and Section VI describes a case study where multiple objects are tracked by the smart camera network. Finally, Section VII concludes the paper.

978-1-4799-5944-0/14/$31.00 ©2014 IEEE

II. Related Work

This section presents related work on dynamic protocol stacks and FPGA-based smart camera networks.

A. Flexible Network Architectures

Already in the early 1990s, Tennenhouse and Wetherall [4] proposed *Active Networks* in which users could inject custom code into the network. This code was associated with a set of packets that traversed the network from the source over several routers to the destination. The code was executed on intermediate nodes and could modify the packets on the fly as desired. A less flexible architecture is proposed by the *Click* modular router [5] and *netgraph* [6]. Both *Click* and *netgraph* offer the possibility to flexibly combine networking functionality. However, they do not focus on run-time reconfiguration.

The concepts of flexibility, modularity, and extensibility were also presented by Ghodsi et al. [7] as the basic requirements for a network architecture that is able to evolve. Wolf et al. [8] argue that a user should be able to choose the service that best fits his requirements. In contrast to related work our dynamic protocol stack architecture adapts the protocol stack to the current environment such as link quality and the applications requirements. Furthermore, our network architecture allows for an adaptive hardware/software mapping of the protocols.

B. FPGA-based Smart Camera Networks

Over the last years it became increasingly popular to implement smart camera platforms partly or entirely on FPGA boards to increase the performance and lower the power consumption as compared to PC-based solutions. Most of the works focused on the image processing and object/person tracking capabilities of the platforms. For instance, Diaz et al. [9] developed a hardware design, which includes reconfigurable window-based image processing elements. The design has been implemented on an Altera Stratix EP1S60 FPGA device that had been connected to a LUPA-4000 image sensor. More recently, Maggiani et al. [10] proposed an architecture which is split into a microcontroller, that processes all network communication, and an FPGA, that implements computer vision algorithms in reconfigurable hardware pipelines.

Similar to our approach, Zarezadeh and Bobda [3] designed an FPGA-based hardware/software architecture for both, the object/person tracking application and the communication middleware. Their smart camera nodes can exchange information using an object request broker. They implemented a prototype on a Xilinx Virtex-4 FPGA, which contains an embedded PowerPC 405. In contrast to our work, their architecture only supports static protocol stacks that are either fully mapped to hardware or fully mapped to software.

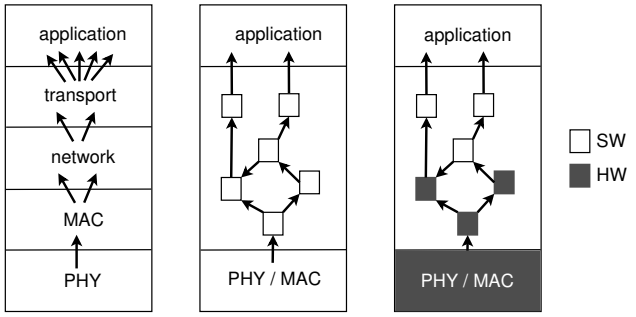
In previous work [11], we have implemented a particle filter-based video object tracking system on a Xilinx Virtex-4 FPGA board. The system-on-chip architecture can change the hardware/software partitioning of the application tasks at run-time in order to meet given performance goals. In contrast to our previous work, we use a different tracking algorithm (a camshift filter [12]), and we extend the scenario from a single camera node to a smart camera network.

III. Background

In order to facilitate the understanding of the smart camera node architecture and the case studies presented in the following sections, we summarize here the DPS and EmbedNet architectures. A detailed description is available in [1].

A. Static vs. Dynamic Protocol Stacks

The current Internet architecture is partitioned in a fixed set of layers and specifies the interaction of a well known set of network protocols, as shown in Figure 1(a). In contrast to the current Internet architecture, protocol stacks can be built dynamically in the DPS architecture.



(a) static PS (b) dynamic PS (c) HW/SW mapping

Fig. 1. Static vs. dynamic protocol stacks (PS). The hardware/software mapping of the used functional blocks can be adapted at run-time.

The basic concept of the DPS architecture is shown in Figure 1(b). The DPS architecture splits network functionality into individual *functional blocks* that process packets on their way from the physical (PHY) / MAC layer to the application layer and vice versa. The functional blocks can be combined at run-time to form a protocol stack, and can be changed on-the-fly in order to provide a communication channel that is continuously optimized.

B. Adaptive Hardware/Software Mapping

We implemented the DPS architecture on a reconfigurable system-on-chip in previous work [1]. In this implementation, called EmbedNet, functional blocks can be implemented either in hardware in the FPGA fabric or in software on the embedded CPU as shown in Figure 1(c). While for performance reasons it would be desirable to implement as many functional blocks in hardware as possible, we are constrained by the FPGA area and by the requirement to include novel protocols that are unknown at design time. Therefore, our EmbedNet prototype dynamically decides which functional block should be mapped to hardware and which to software according to the current processing requirements. In order to dynamically adapt the hardware/software mapping at run time we use the possibility to partially reconfigure modern FPGAs.

IV. Architecture

Our smart camera network consists of a set of smart camera nodes and a controller. The controller offers a user interface that can display the video streams captured by all cameras, and from where the object to be tracked can be selected. Upon the selection of an object, a control message is sent to the

corresponding camera which starts tracking the object. From now on the tracking is performed autonomously by the smart camera network, and the tracking camera simply sends the position of the object back to the controller, where it can be displayed in the user interface. All nodes (smart cameras and controller) are connected by an Ethernet switch. Therefore, the DPS architecture implements a protocol stack that consists of only the Ethernet protocol and omits higher layer protocols such as IP, TCP, or UDP.

The controller is implemented on a commodity workstation as a Linux user-space program. Each smart camera node is implemented as a reconfigurable system-on-chip architecture that is configured on a single FPGA. The hardware/software architecture of a smart camera node is shown in Figure 2. The FPGA design consists of two parts, a video object tracking application (white) and the EmbedNet network node architecture (black). The functionality of both parts is partially mapped to an embedded processor and partially to the FPGA fabric.

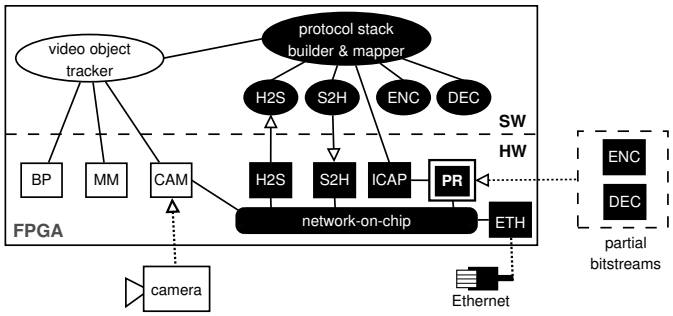


Fig. 2. Hardware/software architecture of a smart camera node: The white part represents the video object tracking application and the black part the EmbedNet network node architecture. The upper half of the FPGA design is mapped to software and the lower half is mapped to hardware.

We have implemented the smart camera node on a Xilinx Virtex-6 ML605 board. A Full-HD Panasonic HC-V727 camcorder is connected to an AVNET HDMI Input/Output FMC module, which is plugged into the ML605 board. The camcorder provides a 1920×1080 video input stream at 50 frames per second (FPS). A Xilinx MicroBlaze processor has been instantiated on the FPGA fabric and runs the software parts of the smart camera node. We have used the ReconOS [13] execution environment in order to seamlessly integrate our hardware blocks to the operating system. ReconOS extends the Linux operating system such that software threads can interact with hardware blocks (or hardware threads) using shared memory and the well-known POSIX interface. The video object tracking application and the EmbedNet architecture can therefore communicate with their hardware blocks. The MicroBlaze processor and all hardware blocks are clocked at 100 MHz.

A. Video Object Tracking Application (white)

The architecture of the video object tracking application on a smart camera node is as follows: The video frames that are captured by the external camera are sent to a hardware module (`CAM`). The `CAM` block scales the input video down to a resolution of 320×180 at 25 FPS. This reduced frame size is then used for all further operations. The `CAM` module sends the captured frames to the software application, where the object tracking is started. We implement object tracking with the help of a camshift filter [12] which requires two computationally expensive operations, namely histogram backprojection (`BP`) and the computation of the moments (`MM`). Therefore, those two blocks are implemented as hardware modules.

To verify if the tracking works correctly, the software application sends the calculated location and size of the object back to the `CAM` module. The `CAM` module can send video frames together with the current location/s of the object/s over the EmbedNet architecture to the controller, which displays the video frames together with the objects position. The video transmission to the controller is optional and can be activated/deactivated by sending a specific control message to the corresponding smart camera node.

B. EmbedNet Architecture (black)

For communication, we use the EmbedNet architecture, which has been already published in previous work [1]. The EmbedNet architecture consists of a *protocol stack builder & mapper* in software, which forms the required protocol stack and dynamically maps the functional blocks either to hardware or to software. The current prototype implementation of EmbedNet supports AES [14] encryption (`ENC`) and decryption (`DEC`) functional blocks in order to establish a secure communication channel. Our current prototype contains one dynamic hardware slot (`PR`), which can be reconfigured at run-time to implement either the `ENC` or the `DEC` block. The partial bitstreams for both blocks are stored in external memory, i.e. SDRAM. The *protocol stack builder & mapper* can reconfigure this slot using the `ICAP` hardware module. The EmbedNet architecture allows to forward packets from software to hardware (`S2H`) and from hardware to software (`H2S`). The `ETH` block can transmit packets to and receive packets from the physical Ethernet interface. In contrast to previous work [1], we extended the hardware/software interface to buffer up to 180 packets in the `S2H` and `H2S` blocks for performance reasons.

The functional blocks in hardware are connected by a network-on-chip (NoC). The NoC consist of three network switches which are connected with each other in a ring topology. Each switch has an interface for the connection to the NoC and two interfaces for connecting functional blocks. In addition to networking functional blocks, the `CAM` module of the video object tracking application is also connected to the NoC. This allows the `CAM` module to directly send packets to the protocol stack, without requiring processor interaction. For the transmission of packets between functional blocks, each packet is extended with a header that specifies the address of the next functional block. If the next functional block is mapped to software, the address corresponds to the `H2S` functional block. The *protocol stack builder & mapper* can configure each functional block with the address of the functional block that should process the packets next.

V. Case Study 1: Single Object Tracking

In this section we describe our first case study where a single object is tracked over the field of view of several autonomous smart camera nodes. This case study shows the correct operation of our smart camera network.

A. Scenario

The smart camera nodes should track a colored toy train, over time. Only one of the smart camera nodes should be in charge of tracking at any given time. As depicted in Figure 3, the cameras are placed next to the oval track, such that each camera sees a different section of the tracks. The train can travel the route clockwise or counterclockwise in different velocities, it can stop in between, and change the direction. The smart camera nodes need to collaborate in order to decide which camera node is in charge of tracking. The fields of view of our smart camera network are shown in Figure 4.

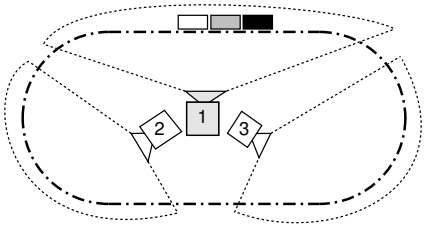


Fig. 3. Case study 1: Tracking a toy train with multiple smart camera nodes.

Initially a user selects the object to be tracked on the controller. The controller computes the histogram of the object (used for the camshift algorithm), and sends it to the smart camera node that currently sees it. This node is initially in charge of tracking. Whenever a smart camera node is tracking the train, it sends the current position and the approximate outline of the train in form of a bounding box to the controller that stores all tracking information. When the object approaches the edges of its field of view a Vickrey auction [15] is performed to determine which camera will continue to track the object. Our handover mechanism has been inspired by Esterle et al. [16].

The node originally tracking the object acts as an auctioneer and sends the object's histogram to the other nodes using a broadcast message. The other smart camera nodes try to find the object in their video input stream using the camshift filter. If a camera node sees the object, it bids for the object by sending a message back to the auctioneer. The height of the bid depends on how well the camera can see the object. The bid value corresponds to the detected size of the object, weighted by the position of the object with respect to the field of view of the camera. The smart camera node with the highest bid wins the auction and is informed by the auctioneer that it should continue tracking. The other nodes are informed that they have lost the auction. It is possible that the auctioneer decides not to handover the tracking responsibility if the bids are low and it can still see the object.

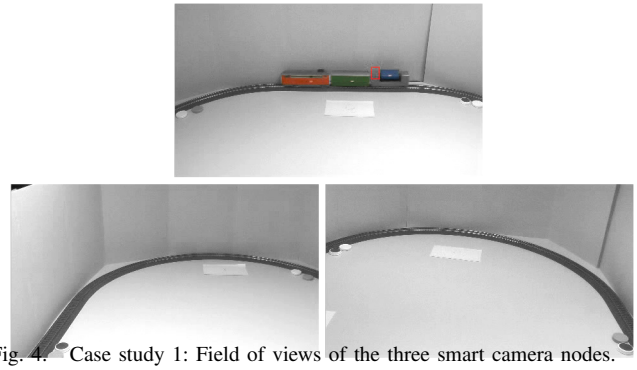


Fig. 4. Case study 1: Field of views of the three smart camera nodes.

B. Results

In order to show the correct hand over between the cameras, we monitor all packets sent by the smart camera nodes. Therefore, we group the packets into two packet types: packet type 1 contains communication between smart camera nodes (auction, handover) and packet type 2 contains the messages from the smart camera nodes to the controller (position and bounding box).

An example measurement for the single-object tracking case study can be seen in Figure 5. In the first 28 seconds, the train runs three circles on the tracks at the lowest speed level, which results in eight handovers. After 28 seconds, the train stops for ten seconds and smart camera node 1 continues tracking for this time interval. Then the train changes its direction and accelerates to the second speed level. The train takes about four more rounds in the final 24 seconds of our experiment, which results in twelve further handovers.

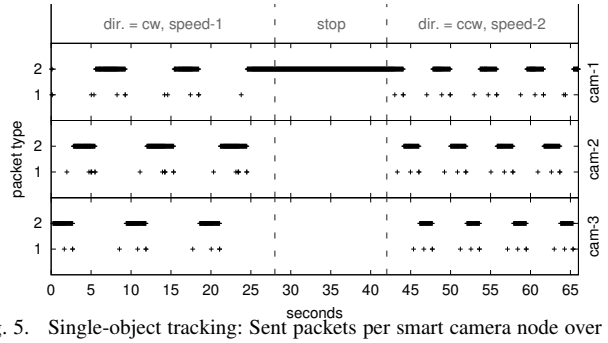


Fig. 5. Single-object tracking: Sent packets per smart camera node over time (packet type 1: tracking handover, packet type 2: tracking result).

This experiment shows that the tracking handover works reliably in our proposed smart camera network in a dynamic scenario, where the train changes its direction from clockwise (cw) to counterclockwise (ccw) and its velocity.

VI. Case Study 2: Multi Object Tracking

In this section we describe our second case study where all network traffic should be encrypted for privacy considerations, and where multiple objects should be tracked. With this case study we show the benefit of using the DPS and the EmbedNet architectures as opposed to the standard Internet architecture for the communication in the smart camera network.

A. Scenario

We assume that a smart camera node can only track a limited number of objects by themselves at the same time. In our case, the camshift filter that is used for tracking only allows for a single object to be tracked. Whenever a node is supposed to track multiple objects, it can transmit its video input stream to (idle) smart camera nodes, which take over the tracking responsibility. We assume that the number of smart camera nodes in the network is at least as high as the number of tracked objects. The selection of the camera that performs the remote tracking works as follows: (i) the camera that is currently responsible for tracking broadcasts a *remote tracking request* to all cameras in the network, (ii) all cameras that are currently not tracking an objects reply, and (iii) the initial camera randomly selects one of the idle cameras for remote

tracking and transfers the histogram of the object to be tracked to this camera. The remote tracking camera then performs the tracking and sends the objects position and bounding box back to the camera that initially was tracking the object.

Figure 6 shows an example with two smart camera nodes and two objects at three different points in time. First, node 1 sees both objects whereas node 2 sees no object in Figure 6(a). In this case, node 1 asks the currently idle node 2 to track one of the objects, i.e. the black ball. If node 2 agrees, node 1 transmits its input video to node 2 over the network. Node 2 then tracks the black object and sends its current position back to node 1. In Figure 6(b) the black object has moved and each node can see a single object. Hence, no video stream has to be transferred. In Figure 6(c) the white object has also moved into the field of view of node 2. Now node 2 streams its video input to node 1, which tracks the white object for node 2.

A video frame is split into L packets, one packet per line. The object's position can be encapsulated into a single packet. Therefore, the ratio between sent and received packets per node is either L:1 or 1:L for a remote video object tracking.

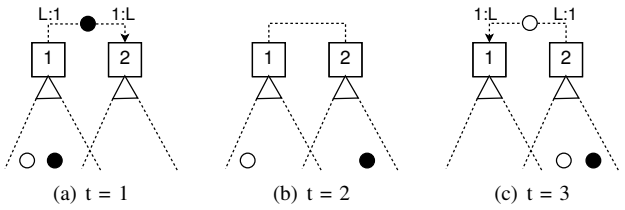


Fig. 6. Case study 2: Tracking two objects with two smart camera nodes. (a) Node 1 sees two objects and transmits its video stream to node 2, (b) both nodes see and track one object, and (c) node 2 sees two objects and streams its video to node 1.

For privacy reasons, we require a secure connection between all nodes since the video streams and the tracking results might contain sensible data. Therefore, we insert an AES encryption block for outgoing packets and an AES decryption block for incoming packets into the protocol stacks. Since packet encryption/decryption is computational expensive, it would be desirable if this could be accelerated in hardware. However, our FPGA-based system-on-chip platform only offers the space for one hardware accelerator, e.g., for either an encryption, or a decryption block, but not for both. Hence, in order to achieve the best tracking performance, a smart camera node needs to dynamically adapt its hardware/software mapping of functional blocks over time, whenever it changes its role from transmitting a video stream to receiving a video stream (and vice versa).

B. Results

For this scenario we evaluated the packet throughput of the protocol stack for different hardware / software mappings of the encryption and decryption functional blocks, then we analyzed the FPGA resources required by our smart camera nodes, and finally we quantified the overhead introduced by the adaptive architecture.

1) Hardware/Software Mappings: We analyzed the packet throughput supported by a single smart camera node for several hardware / software mappings of the encryption and decryption blocks. Table I summarizes our results. If the node transmitting the video implements the encryption block in hardware, it is able to transfer at the requested rate of 25 FPS. However, if it implements the encryption block in software, it can only transmit two FPS. Similarly, for the node receiving the video: If it implements the decryption block in hardware, it can receive 25 FPS, but if it implements the decryption block in software it can only receive two FPS. In Table I, we also see that the receiving node looses about 0.3% of the packets, even if the decryption module is implemented in hardware. This packet loss is introduced by the hardware / software interface, which has a limited bandwidth. However, the video object tracking application is robust to a certain amount of packet loss. Whenever a video packet is lost, the application will use the pixel line of the previous frame instead.

TABLE I. MEASUREMENT FOR PACKET THROUGHPUT IN PPS, THE PACKET LOSS RATE IN % AND THE VIDEO TRANSMISSION RATE IN FPS.

send (encrypted) frames	**PPS**	**Loss**	**FPS**
HW:CAM → H2S → **SW:ENC** → S2H → HW:ETH	429	90%	**2**
HW:CAM → **HW:ENC** → HW:ETH	4 500	0%	**25**
receive and decrypt frames	**PPS**	**Loss**	**FPS**
HW:ETH → H2S → **SW:DEC** → tracking application	420	91%	**2**
HW:ETH → **HW:DEC** → H2S → tracking application	4 487	0.3%	**25**

In the multi object tracking scenario not only video data is transmitted, but also control data (e.g. object selection, handover, and object position messages). These messages are encrypted as well and might flow in the opposite direction of the video data transmission. Therefore, a smart camera node might have to provide encryption and decryption at the same time. Control messages are expected to be way less frequent than video data messages, since for each received video frame (consisting of 180 packets), only one control message is sent back. Object selection and handover messages will occur even less frequently. From Table I we see that a node that encrypts in hardware can decrypt about 400 packets per second in software (and vice versa), which should be more than enough to process all control messages.

From this analysis we learn that for successfully tracking multiple objects in our smart camera network, the smart camera nodes need to adapt their hardware / software mapping of the encryption / decryption modules, depending on whether they currently transmit or receive the video frames.

2) Resource Consumption: The total resource consumption of our adaptive architecture and different static architectures is given in Table II. It can be seen that the adaptive architecture with one reconfigurable hardware slot consumes a slightly higher amount of look-up tables (LUTs) and flip-flops (FFs) as the static architectures that contain either `HW:ENC` or `HW:DEC`. This comes mainly from the overhead introduced by the ICAP controller that is needed to perform the partial reconfiguration of the FPGA for the adaptive system. The ReconOS ICAP controller used in our implementation consumes 5% of the resources of an encryption/decryption block.

We have also implemented a static architecture that contains both hardware blocks, which results in 25% more LUTs and 22% more FFs as compared to the adaptive architecture. It should be noted that an adaptive architecture can provide much larger savings in resources when there are more functional blocks that can be dynamically mapped to the FPGA fabric.

TABLE II. RESOURCE CONSUMPTION

mapping	LUTs	FFs	BRAMs
static: HW:ENC	34 297	25 318	314
static: HW:DEC	36 041	25 653	314
adaptive: HW:ENC or HW:DEC	**37 534**	**25 949**	**314**
static: HW:ENC and HW:DEC	46 743	31 689	314
ReconOS ICAP module + FIFOs	690	255	-
partial module: HW:ENC	11 974	6 648	-
partial module: HW:DEC	14 165	6 983	-

3) Overhead Analysis: In addition to the slightly increased FPGA resources required for the adaptive system, also a time overhead for the reconfiguration as well as the need to store the partial bitstreams for the encryption and decryption blocks need to be considered. In our implementation the partial bitstreams have a size of 1.3 MB and the time overhead is 38ms. This corresponds to the time that is required to process a single video frame. The time overhead consists of the following parts: First, the *protocol stack builder & mapper* maps all functional blocks to software. Second, the FPGA is partially reconfigured with the bitstream of the functional block that should be mapped to hardware. Third, the *protocol stack builder & mapper* configures the new hardware functional block (in our example with the encryption key) and adapts the protocol stacks to use the new hardware functional block.

To summarize, the static architecture that implements both `HW:ENC` and `HW:DEC` would provide the best packet processing performance because this architecture allows for a video transmission of 25 FPS without requiring any reconfiguration overhead. However, we believe that it is not always feasible to statically map all functional blocks to hardware since the FPGA resources are usually limited. Therefore, we need efficient strategies to dynamically switch between different hardware blocks at run-time (according to the current protocol stacks and network traffic mix). Our experimental results show that the HW/SW mapping of our adaptive architecture can be efficiently updated for a given tracking scenario such that the packet processing performance is only influenced marginally.

VII. CONCLUSION

In this paper we proposed a novel system-on-chip architecture for smart camera nodes, where the application and the networking core are both partitioned into hardware and software. The smart camera architecture employs dynamic protocol stacks (DPS) instead of static protocol stacks such as UDP/IP or TCP/IP. The DPS architecture allows to adapt the hardware/software partitioning of the protocol stacks at run-time. We analyzed the efficiency of our smart camera node architecture in two case studies.

First, we demonstrated that our smart camera network is able to autonomously track a toy train over the field of view of three smart cameras when the train changes its direction and its velocity. Second, we showed that it is beneficial (in terms of required FPGA resources) when the network functionality can be dynamically mapped to either hardware or software. We also demonstrated that this re-mapping can be performed within the delay of a single video frame. Hence, we conclude that the dynamic hardware / software mapping of network functionalities is a real alternative to static designs.

In future work, we want to extend our smart camera nodes such that they can autonomously adapt the protocol stacks (and not only the corresponding hardware/software mapping) at run-time. For instance, the nodes could dynamically require packet acknowledgements for all auction and handover messages whenever a video stream is transmitted over the network. Furthermore, we want to develop and analyze smart protocol stack adaptation and mapping strategies which find suitable trade-offs between the benefits and costs of run-time adaptation. Finally, we plan to integrate further functional blocks such that we can build more complex protocol stacks.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n°257906. The authors want to thank Jan Krekeler, Andreas Agne and Marco Platzner for providing the implementation of the camshift filter.

REFERENCES

[1] A. Keller, D. Borkmann, S. Neuhaus, and M. Happe, "Self-awareness in Computer Networks," *Hindawi International Journal of Reconfigurable Computing*, 2014.

[2] B. Rinner and W. Wolf, "An Introduction to Distributed Smart Cameras," *Proceedings of the IEEE*, vol. 96, no. 10, Oct 2008.

[3] A. A. Zarezadeh and C. Bobda, "Hardware Middleware for Person Tracking on Embedded Distributed Smart Cameras," *Hindawi International Journal of Reconfigurable Computing*, Jan 2012.

[4] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 5, 2007.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, 2000.

[6] "netgraph – Graph-based Kernel Networking Subsystem," (accessed in Oct. 2014). [Online]. Available: {http://www.freebsd.org/cgi/man.cgi?query=netgraph&sektion=4}

[7] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Intelligent Design Enables Architectural Evolution," in *Workshop on Hot Topics in Networks*, ser. HotNets-X. ACM, 2011.

[8] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldine, and A. Nagurney, "Choice as a Principle in Network Architecture," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, 2012.

[9] F. Dias, F. Berry, J. Serot, and F. Marmoiton, "Hardware, Design and Implementation Issues on a FPGA-based Smart Camera," in *Int. Conf. on Distributed Smart Cameras (ICDSC)*. ACM/IEEE, 2007.

[10] L. Maggiani, C. Salvadori, M. Petracca, P. Pagano, and R. Saletti, "Reconfigurable FPGA Architecture for Computer Vision Applications in Smart Camera Networks," in *Int. Conf. on Distributed Smart Cameras (ICDSC)*. ACM/IEEE, 2013.

[11] M. Happe, E. Lübbers, and M. Platzner, "A Self-adaptive Heterogeneous Multi-core Architecture for Embedded Real-time Video Object Tracking," *Journal of Real-time Image Processing*, vol. 8, March 2013.

[12] X. Chen, X. Li, H. Wu, and T. Qiu, "Real-time Object Tracking via CamShift-based Robust Framework," in *Int. Conf. on Information Science and Technology (ICIST)*. IEEE, 2012.

[13] A. Agne, M. Happe, A. Keller, E. Lübbers, B. Plattner, M. Platzner, and C. Plessl, "ReconOS – An Operating System Approach for Reconfigurable Computing," *IEEE Micro*, pp. 60–71, Jan/Feb 2014.

[14] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[15] W. Vickrey, "Counterspeculation, Auctions, and Competitive Sealed Tenders," *Journal of Finance*, vol. 16, pp. 8–37, Mar 1961.

[16] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner, "Socio-Economic Vision Graph Generation and Handover in Distributed Smart Camera Networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 2, 2013.