Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2015) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

10th International Conference on Future Networks and Communications, FNC-2015 and the 12th International Conference on Mobile Systems and Pervasive Computing, MobiSPC 2015

Formalization of the Behavior of Content-Centric Networking

Sosuke Moriguchi[a,*], Takashi Morishima[a], Mizuki Goto[a], Kazuko Takahashi[a]

[a]*Kwansei Gakuin University, 2-1, Gakuen, Sanda, 669-1337, JAPAN*

Abstract

Content-Centric Networking (CCN in short) is a communication architecture which is based on the name of contents, rather than on addresses. A protocol used in CCN is not for End-to-End communications, but for network-wide communications. Each node sends packets to connecting nodes, and these nodes communicate other nodes connecting with them. When data are sent, the receiving node stores it and forwarding other nodes. Such stores increase reliability of data and divide loads for servers. However, the behaviors and performances of the protocol are under investigation. In this paper, we formalize the CCN protocol using proof assistant Coq. We create two parts of formalization, the network module type and the behaviors of the protocol. The network module type has several parameters such as an data type denoting nodes, connection relations between them, and some status depending on CCN. Then we prove two specifications about content-deliveries in CCN. With a specific network description based on the module type, we get the proofs of these specifications on the given network directly. This result can be used to enhance the reliability of CCN protocols.

Keywords: Content-Centric Networking, Protocol Verification, Coq

1. Introduction

With the growing trend of cloud computing, users commonly know only that their data exist somewhere in a network; they tend to be interested only in fast and safe retrieval, and the location of the data is unimportant. With internet protocol (IP), communication is undertaken by assigning an address name to a server, i.e., where a content is located. When a user wants to access some data, they make an access request to the server and can access the content only if the request arrives at the server. However, the user typically does not care about the physical location of contents, but rather is interested only in the content. The Content-Centric Network (CCN) was developed[1] as an alternative network architecture that focuses not on "where" but on "what". In CCN, communication is undertaken based on the name of the content, rather than the location thereof. The fundamental idea of CCN is broadcasting and storing information embedded at each node. End-to-end communication is not required and the user can retrieve data from a closer node by matching with the stored information. As a result, the efficiency and the reliability of network can be increased, and the response time can be reduced. Although prototype implementations of this architecture

[*] Corresponding author. Tel.: +81-79-565-8391.
E-mail address: chiguri@acm.org

have been developed, the technique is still under development. Many significant issues remain, including security, congestion, and deadlock. For example, security vulnerabilities have been pointed out[2], and a revised protocol was proposed to cover the drawbacks of flooding[3]. Correctness of the behavior has not been guaranteed, either.

It is difficult to show behavioral correctness over a network, because it requires checking all possible cases with regards to the network topology, data structure, and timing of communication. Correctness can be checked using test and runtime verification; however, it is not complete. So far, in the development of network systems, emphasis has been placed on network performance, and strict correctness of the behavior has not been required. It has been considered sufficient if communications succeed at a high probability; even if it may fail or lock, whereupon timeout and retry mechanisms can be used to solve the problem. However, a single error may result in a significant failure that is difficult to recover from in a large, complicated network. Therefore, it is considerably important to certify the correct behavior of protocols.

Wang emphasized the importance of providing a sound network design, and proposed declarative networking and protocol verification using a proof assistant[4]. Proof assistants represent a formal method to develop a certified system. There have been several reports of the formalization of network protocols using proof assistants[5], and primarily issues with routing have been considered, without considering the contents of packets. On the other hand, as CCN does not employ end-to-end communication, we must describe the treatment of the contents of packets as well as the routing. An alternative formal method is model checking[6], and there have been several reports of the use of model checkers on verification of network protocols[7,8]. However, there exists a state explosion problem in model checking.

In this paper, we aim at formalizing CCN and showing behavioral correctness using the proof assistant Coq[9]. In CCN, a node stores data on the nodes from which messages are received, and so may cache previously received data. The node looks up the stored information and checks whether any matching data exist. If this data matching is successful, the node returns the cached data to the requester. This matching is carried out using the content name, and the request does not always have to go to the server. We define sending a packet and updating the stored information as events, and create an inductive model using a sequence of events. For behavioral correctness, we take the property that a node can retrieve a content if and only if the user sends a request, when the content exists in the network. There are several possible routes, because a request may be sent to any node on which contents are cached. We prove correctness using induction on a sequence of events for all possible cases. These cases are tangled and difficult to check exhaustively by hand. Coq will check such exhaustiveness mechanically and tell the rest of cases if exists. This is the first attempt to give a formalization of CCN protocol using a proof assistant.

We describe temporal properties such as eventuality and causality using a sequence of events. This is a different approach from that typically employed in model checking, in which the entire system is modeled as a state transition system. We solve the problem of state explosion using an induction scheme with the proof assistant. Note that the transition system does not consider performance of the behaviors[1].

The rest of this paper is organized as follows.

- We explain CCN in section 2.
- In section 3 and section 4, we formalize a network and behaviors of the protocol, respectively.
- Section 5 shows two specifications for CCN and introductions of their proofs.
- We show related work in section 6 and summarize in section 7.

In this paper, we explain formalization of CCN using Coq, but we will show very limited parts of scripts due to a space constraint. You can see whole scripts of the formalization and proofs at `https://github.com/chiguri/CCNprotocol`.

2. Content-Centric Networking

Here, we explain the CCN protocol using an example. In recent network protocols such as HTTP, there are three kinds of nodes, users sending requests, servers receiving (and replying) requests, and rooters forwarding requests. In

[1] Rigorously, the system is not runnable since it is defined as predicates on a higher-order logic.

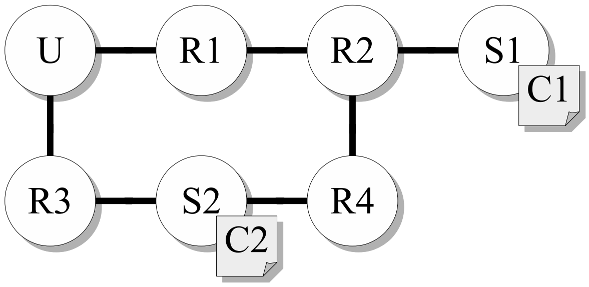

Fig. 1: Simple network with 7 nodes.

Table 1: FIBs in Fig. 1

Content-Name	From	To	Content-Name	From	To
`c1`	`R1`	`R2`	`c2`	`R1`	`R2`
	`R2`	`S1`		`R2`	`R4`
				`R3`	`S2`
	`R4`	`R2`		`R4`	`S2`

contrast, in CCN, each node can be a user, a rooter and/or a content-server. Here, we use these words for a requesting node, a node forwarding packets and a node retaining some contents from the beginning, respectively. This section is basically the same with the original paper[1], but we abstract some concrete parts which do not matter the formalization.

Assume that a network is shown in Fig. 1. There are a user `U`, rooters `R1` to `R4`, and servers `S1` and `S2`. We have two contents `C1` and `C2` in the network, stored in `S1` and `S2` respectively. `C1` is named `c1` and `C2` is `c2`.

In the case `U` wants to access `C1`, `U` sends *interest packets* to all connecting nodes, `R1` and `R3`, and waits *data packets*. Interest packets contain content-names such as `c1` and `c2`[2], and data packets contain content-names and contents. In the CCN protocol, we use only interest/data packets to send and receive requests/data.

`R1` receives an interest packet with `c1`, it checks it has `C1` or not. If it does not have `C1`, then it checks its *Forwarding Information Base (FIB)*. FIB is a mapping from content-names to lists of nodes for *forwarding interest packets*. Let FIBs for the nodes be shown in Table 1. If there is no mapping from given content-names, interest packets with them are dropped. In this case, `R1` forwards the packet to `R2`. Also, `R1` stores the node (`U`) from where the packet is sent in its *Pending Interest Table (PIT)* . `R3` also receives the interest packet with `c1`, but its FIB does not have mapping from `c1`, thus `R3` drops the packet.

After `R2` forwarding an interest packet with `c1` in the same manner, `S1` receives an interest packet with `c1`. In this case, `S1` replies with a *data packet* containing `C1` to `R2` rather than forwarding the packet or dropping it.

When `R2` receives a data packet, it checks whether it already has a content. If it has, the packet is dropped. If it does not have, it searches PIT for the content. In this example, `R2` has an entry for `c1` (sent from `R1`), so `R2` forwards the packet to `R1`. At the same time, `R2` stores the content. After storing the content, if `R2` receives interest packets with `c1`, it replies directly with data packets containing `C1` as if it is a content-servers. When `U` receives a data packet containing `C1`, it stores the packet since it *requested* the content.

In the case `U` wants `C2` (initially served by `S2`), there are two network path from `U` to `S2` through FIBs, `R3` to `S2` and `R1`, `R2`, `R4` to `S2`. We cannot decide from which path `U` will receive the content, and the both cases are valid in CCN.

[2] In the original CCN protocol, a content-name is in the path format such as `/foo/bar/music.mp3`. In this paper, we omit such details.

3. Model of Networks

We divide formalization of the CCN protocol to two parts; one is a network model and the other is a behavioral model based on the network. This section shows the network model, whose topology is parameterized.

The model has some parameters and assumptions for the topology. More specifically, parameters are about nodes, connections, initial content-servers and FIB.

Node. First of all, all nodes should be identified by a data type. We require all nodes can be distinguishable with each other.

Connection. For each node, all nodes connecting the node should enumerable as lists. We define it as a function from nodes to lists of nodes. Additionally, connections should be symmetric, i.e., if node n_1 connects with node n_2, then n_2 should connect with n_1 also.

Initial Content-Server. Some nodes should have contents initially. In the formalization, we require decidability whether a node has a specific content or not.

Forward Information Base. FIB is formalized as a function from node to node lists, like connections. Also, we require that nodes in FIB should connect, i.e., if node n_1 is in node n_2's FIB list, then n_1 and n_2 should connect each other. This feature forces forwarded packets are transfered between connecting nodes. If a node does not have FIBs for a content-name, the returned list for the node and the content-name should be empty.

. We have predicate over nodes and content-names, named `FIBreachable`, which shows a given node is reachable at some initial content-servers for a given content-name by tracing FIB. This ensures that the node is *accessible* to the data in the network.

If we define the module denoting the network in Fig. 1, we may define the nodes as an inductive data type in Coq.

```
Inductive Nodes : Set := | U | R1 | R2 | R3 | R4 | S1 | S2.
```

The following code denotes connected nodes for a given node.

```
Definition Connected_list (v : Nodes) : list Nodes :=
match v with
| U  => [R1; R3]
| R1 => [U ; R2]
| R2 => [R1; S1; R4]
| R3 => [U ; S2]
| R4 => [R2; S2]
| S1 => [R2]
| S2 => [R3; R4]
end.
```

In the similar manner, we define content-names (`c1` and `c2`) and FIBs for each node.

```
Inductive Content_Names : Set := c1 | c2.

Definition FIB_list (v : Nodes) (c : Content_Names) : list Nodes :=
match c with
| c1 => match v with
        | R1 => [R2]
        | R2 => [S1]
        | R4 => [R2]
        | _  => []
        end
| c2 => match v with
        | R1 => [R2]
        | R2 => [R4]
        | R3 => [S2]
        | R4 => [S2]
        | _  => []
        end
end.
```

If `FIB_list` with a node and a content name returns an empty list, it means the node does not have any FIBs for the content.

As we explained before, we have to prove some restrictions about nodes, connection relations, etc. However, in the case of the simple network, these restrictions are straightforward and easy to prove. When we make a module named `CCN_SimpleNetwork` containing the previous definitions, Coq checks whether required parameters are in the module. If some restrictions are not shown, Coq does not allow to define the module.

4. Behaviors of Protocols

In CCN, we can observe some actions and some packets produced by actions. We say such actions *events* hereafter. An event is a request of a content (`Request` in the formalization), forwarding an interest packet (`ForwardInterest`), adding a node into PIT (`AddPIT`), replying with a data packet (`ReplyData`), forwarding a data packet (`ForwardData`), or storing a content in given data packet (`StoreData`).

We formalize the behaviors of a given network as a sequence of events and a list of packets. They denote all occurred events sorted in chronological order and all sent but unprocessed packets, respectively. This means that the formalization of the protocol is concurrent rather than parallel. In this paper, we call a pair of the event sequence and the packet list *a snapshot*.

There are no explicit PITs nor stored contents in a snapshot, but we can calculate them from the event sequence by picking up `AddPIT` and `StoreData` events. In every time we need them, we use functions calculating them in the formalization.

The behaviors of the protocol are defined as follows. In any time, every node can request a content if it does not have the content. Other behaviors are based on the snapshot.

- If there exists an unprocessed interest packet, and a node has a requested content, then it replies with a data packet containing the content.
- If there exists an unprocessed interest packet, but no FIB is available for a content-name contained in the packet, then a node drops the packet.
- If there exists an unprocessed interest packet, but no PIT entries for a content-name contained in the packet, then a node forwards the packet to nodes in FIB and add an PIT entry.
- If there exist an unprocessed interest packet and PIT entries for a content-name contained in the packet, then a node adds an PIT entry but does not forward the packet.
- If there exists an unprocessed data packet, but a node has a corresponding content, then it drops the packet.
- If there exist an unprocessed data packet and PIT entries for a corresponding content-name, then it stores the content and forwarding the packet.
- If there exists an unprocessed data packet and a node requested a corresponding content, then it stores the content.
- If there exists an unprocessed data packet, but a node did not request a corresponding content and there are no PIT entries for the content-name, then it drops the content.

These unprocessed packets are selected nondeterministically. We do not restrict orders of processing packets. This means that the behaviors are completely asynchronous.

In Coq, we define an inductive predicate that a snapshot complies with the CCN protocol as `CCNprotocol`. The predicate is satisfied with an initial state which is an empty event sequence and an empty list, and a snapshot generated by the behavior described above from another snapshot satisfying `CCNprotocol`. Since `CCNprotocol` is an inductive predicate, we can proof by induction on the behaviors if a given snapshot complies with the protocol.

5. Specifications and Proofs

We prove two specifications for the CCN protocol as behavioral correctness. One is similar to eventuality of content-delivers, and the other is causality of arrivals of the contents. We call the former *forward lemma* and the latter *backward lemma*. Before proving them, we show any packets in snapshots complying with the CCN protocol are sent between connecting nodes.

Proofs for the forward/backward lemma are constructed on parameters describing networks, but not on implementations of networks. We can get proofs for forward/backward lemmas by applying the proof module, named `CCN_Protocol_Verification` to the network module describing a network you intend. In other words, we prove forward/backward lemmas of the CCN protocol in *any networks we can describe in Coq*, e.g. fixed number of nodes or unbound number of nodes with inductive topologies. For example, for the simple network described in section 3, the proofs are defined with the following command.

```
Module CCN_SimpleNetwork_Verification := CCN_Protocol_Verification CCN_SimpleNetwork.
Import CCN_SimpleNetwork_Verification.
```

Networks with fixed number of nodes are very easy to describe in Coq since we can enumerate the nodes as inductive data types. We also define networks for unbound number of nodes configurating half line and binary tree.

5.1. Forward

If we have requested some contents, we expect that we eventually get contents. However, without strong restrictions, we cannot prove such eventuality.

Since each node behaves concurrently, there is an event sequence such that some nodes send interest packets ceaselessly. In the case, requests from others may be abandoned and *never* processed. This is not caused by the formalization since in general we cannot assure that *any requests will be processed properly*.

To avoid such problems, we define the specification of forward lemma as follows.

```
Theorem CCN_Forward_Request :
 forall (v : Node) (c : Content_Name) (es : list Event) (ps : list Packet),
 (exists v' : Node, Connected v v' /\ FIBreachable v' c) ->
  CCNprotocol (Request v c :: es) ps ->
   forall (es' : list Event) (ps' : list Packet),
   CCNprotocol (es' ++ Request v c :: es) ps' ->
    (exists C : Content c,
     In (StoreData v c C) (es' ++ Request v c :: es))
     \/ (exists (C : Content c) (es'' : list Event) (ps'' : list Packet),
          CCNprotocol (StoreData v c C :: es'' ++ es' ++ Request v c :: es) ps'').
```

The specification says that, after a node requests contents, the network snapshot is always under following states:

1. the node already received the content (i.e., `StoreData` is in the event sequence), or
2. the node *may* receive the content after some times.

We split the specification into two parts, one is for users requesting contents, and the other is for rooters forwarding interest packets. The specification for users is the same as the above except that `Requests` are `ForwardInterests`. First we prove the specification for rooters and then that for users. The proof for rooters is done by induction on the `FIBreachable` predicate and case analysis for the snapshots. It is about 140 lines of codes to prove, and lemmas proved by around 1000 lines of codes. The proof for users uses case analysis and the previous specification. It is also about 140 lines of codes. Since the case analysis is very complex, without proof assistant, we might miss some cases.

5.2. Backward

The backward lemma says that if a node receives data packets, then it sent interest packets (as forwarding packets or requesting contents) before it receives. This kind of causality is possible to describe directly with past-time operators[10]. Such operators are known to be describable with usual temporal logic such as CTL and LTL, but it is far from succinct nor intuitive[6]. In contrast, we can describe it without any extra operators.

```
Theorem CCN_Backward :
 forall (v : Node) (c : Content_Name) (C : Content c) (es : list Event)
                                                      (ps : list Packet),
  CCNprotocol (StoreData v c C :: es) ps ->
   In (Request v c) es \/ In (ForwardInterest v c) es.
```

The proof is done by induction on the protocol behaviors. It is about 10 lines of (small automated) codes to prove, and one lemma (3 lines to prove).

6. Related Work

The formalization of behavioral correctness of a network protocol, such as routing, forwarding, and addressing, has also been well studied. Karsten et al. showed meta-level axiomatization for forwarding using Hoare-style logic[11]. However, their proof was carried out using a pen-and-paper approach. Felty et al. described a scalable coherent interface (SCI) cache coherence protocol using Nurpl[12]. Bharadwaj et al. gave an inductive formalization for routing algorithms and proved their correctness using Coq and the model checker SPIN[13]. Bhargavan et al. formalized an ad-hoc network protocol and proved its behavioral correctness using HOL and SPIN[5]. Stewart showed a verification of declarative network programs using Coq[14].

In most of these previous reports, the authors focused on the correctness of routing. However, here we focus on the content of packets, because CCN employs content-based communication.

An alternative approach to proving behavioral correctness is the use of a model checker[6]. The required specification can be represented using temporal logic, because there are 'always' and 'eventuality' operators. There have been a number of attempts to verify networks using model checkers[7,8]. Reachability and memory leak were analyzed in these works. However, their main goal was bug detection, whereas our goal is to give a proof of the correctness of the behavior of the network. Moreover, there exists a state explosion problem in model checking, whereas this can be avoided through the use of induction via a proof assistant. Also, we have parameterized network topologies, but it is hard to represent in model checking.

We have formalized the causality of events in a proliferating sequence of events in both the forward and backward directions. It follows that we can specify the property "if users send a message in any situation, then they eventually receive the corresponding answer" and "if users receive a message, then they sent the corresponding request" without expressing a time explicitly.

7. Conclusion

We formalized the CCN protocol using proof assistant Coq. We created an inductive model using a sequence of events and showed forward validity and backward validity. Although we used Coq, this method can be applied using many other proof assistants.

The main contributions of this work are twofold: (1) This represents the first attempt to give a formalization of a CCN protocol using a proof assistant. This can significantly enhance the reliability of a CCN protocol. (2) We have shown the formalization of causality of events in a proliferating sequence of events, in both the forward and backward directions. This represents an extension of the application areas of proof assistants.

In the future, we plan to extend the protocol with limiting a number of contents stored in a node. We believe such limitations do not break the specifications proven in this paper. In contrast, we already know that limitations of numbers of PITs break the forward-specification because a request may drop out of PITs. To keep the specification, we need other functions such as resending requests. Since they are not protocol requirements, we should discuss which functions are better for the protocol.

References

1. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.. Networking named content. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*; CoNEXT '09. New York, NY, USA: ACM. ISBN 978-1-60558-636-6; 2009, p. 1–12. doi:10.1145/1658939.1658941.
2. Goergen, D., Cholez, T., François, J., Engel, T.. Security monitoring for content-centric networking. In: Pietro, R.D., Herranz, J., Damiani, E., State, R., editors. *DPM/SETOP*; vol. 7731 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-642-35889-0; 978-3-642-35890-6; 2012, p. 274–286.

3. Carofiglio, G., Gallo, M., Muscariello, L.. ICP: Design and evaluation of an interest control protocol for content-centric networking. In: *INFOCOM Workshops*. IEEE. ISBN 978-1-4673-1016-1; 2012, p. 304–309.
4. Wang, A., Loo, B.T., Liu, C., Sokolsky, O., Basu, P.. A theorem proving approach towards declarative networking. In: *22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009)*. 2009.
5. Bhargavan, , Obradovic, , Gunter, . Formal verification of standards for distance vector routing protocols. *Journal of the ACM* 2002;**49**.
6. Clarke, E.M., Grumberg, O., Peled, D.A.. *Model Checking*. Cambridge, Massachusetts: The MIT Press; 1999. ISBN 0262032708.
7. Killian, C., Anderson, J.W., Jhala, R., Vahdat, A.. Life, death, and the critical transition: Finding liveness bugs in systems code. In: *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation*. 2007, p. 243–256.
8. Musuvathi, M., Engler, D.R.. Model-checking large network protocol implementations. In: *First symposium on Networked Systems Design and Implementation*. 2004, p. 155–168.
9. Coq. The Coq proof assistant. `http://coq.inria.fr/`; [Last access: March 24, 2015]
10. Gabbay, D.. The declarative past and imperative future. In: *Temporal logic in specification*. Springer; 1989, p. 409–448.
11. Karsten, M., Keshav, S., Prasad, S., Beg, M.. An axiomatic basis for communication. In: *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*; SIGCOMM '07. ACM. ISBN 978-1-59593-713-1; 2007, p. 217–228. doi:`10.1145/1282380.1282405`.
12. Felty, A.P., Howe, D.J., Stomp, F.A.. Protocol verification in nuprl. In: *10th International Computer Aided Verification Conference*. 1998, p. 428–439.
13. Bharadwaj, R., Felty, A.P., Stomp, F.A.. Formalizing inductive proofs of network algorithms. In: Kanchanasut, K., Levy, J.J., editors. *Algorithms, Concurrency and Knowledge: 1995 Asian Computing Science Conference, ACSC '95*; vol. 1023 of *Lecture Notes in Computer Science (LNCS)*. Pathumthani, Thailand: Springer; 1995, p. 335–349.
14. Stewart, G.. Computational verification of network programs in coq. In: Gonthier, G., Norrish, M., editors. *Third International Conference on Certified Programs and Proofs (CPP 2013)*; vol. 8307 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-03544-4; 2013, p. 33–49.