A Methodology for Model-checking Ad-hoc Networks

Irfan Zakiuddin[1], Michael Goldsmith[2,3], Paul Whittaker[2], and Paul Gardiner

[1] QinetiQ, Malvern, UK,
I.Zakiuddin@eris.QinetiQ.com,
[2] Formal Systems (Europe) Ltd.
{michael,paulw}@fsel.com,
WWW home page: http://www.formal.demon.co.uk
[3] Worcester College, University of Oxford

Abstract. Wireless networks, specifically *ad-hoc* networks, are characterised by rapidly changing network topologies. Their dynamic nature makes it a challenge to design and assess protocols for such networks. We present a methodology, based on CSP and the FDR model-checker, to validate critical properties of such networks, properties like self-stabilisation. Our work started by applying CSP/FDR to a tactical internet (a military mobile network). The techniques developed there were generalised to our methodology for model-checking ad-hoc networks and more general self-configuring systems. We first give an overview of the results of model-checking the tactical internet, then we describe the methodology on an ad-hoc network case study, namely the Cluster-Based Routing Protocol. The methodology is quite generic, but it enables the complex dynamic properties of ad-hoc networks to be captured quickly and easily, in models that are ususally readily tractable. We end with a brief discussion of some of its other applications.

1 Introduction

The ARPANET bug [1] showed how critical it is to design correct networking protocols. There, one faulty router issued a few corrupt topology update messages before crashing, and that was sufficient to livelock the entire ARPANET. To break the livelock, every single router in the ARPANET had to be manually re-booted. The problem was that the network management protocols in the ARPANET were not *self-stabilising*. Perlman [1] defines a network as self-stabilising when, after some fault, it is able to return to a normal state in a 'reasonable' amount of time, without human intervention. She also requires that the fault does not recur. Perlman's notion of self-stabilisation is related to Dijkstra's [2], but is more informal and focused on networking protocols. Her more pragmatic notion served as a useful starting point for our investigations.

Modern networks are increasingly mobile, using wireless communications. An extreme class are *M*obile *A*d-hoc *NET*works, MANETs, [3], which do not even rely on a fixed infrastructure to serve nodes that are potentially moving very fast. Rather, a MANET is required to form and maintain itself spontaneously. Self-stabilisation for these networks is particularly challenging. Protocols for MANETs are prone to get caught in a cycle of perpetual self-configuration - even in the absence of faults, and merely as a consequence of a dynamic topology.

This paper describes work done at QinetiQ (formerly part of DERA) and Formal Systems (Europe) Ltd, to apply CSP and FDR [4] to study protocols for self-configuring networks. The driving case study for that work was a 'tactical internet'; in effect this is a MANET for deployment in the theatre of battle. Studying the tactical internet produced techniques of significant interest and value. However, the tactical internet is commercially confidential (not to mention its military sensitivity!). To further advance the techniques and to enable their exposure we applied them to some public domain systems, viz:

- the Cluster Based Routing Protocol (CBRP) [5], which is a MANET
- Mobile IP

These studies yielded, in effect, a methodology for model-checking ad-hoc networks, and more generally for self-configuring systems. Further use was made of this methodology in our wider research work on:

- a self configuring key hierarchy, for group key management, [6]
- a link reversal routing algorithm, for MANETs.

And these various items confirmed to us the value and the interest of our approach. The methodology derives its power from flexible and simple concepts, which are supported by CSP_M's powerful programming capabilities.

Of course our models are limited to a small finite number of principals, typically about 5, but this is often sufficient locate undesirable behaviour. To *verify* systems like these, with a model-checker, requires some form of inductive reasoning. In a CSP and FDR context, they will typically be based on data independent induction, [7]. Techniques for data independent induction establish the base and step case for infinitely many inductions, at the same time. In effect, results are proved for an unbounded number of principals.

The methodology we present is intended as a precursor to the use of data independent induction. It enables the complex interactive behaviour of these dynamic systems to be captured in models that are tractable and easy to create. These models can then be the basis for subsequent inductive reasoning.

The rest of this paper presents an overview of modelling the tactical internet, and the results achieved. We then discuss the general methodology before illustrating it with its application to CBRP. The conclusion discusses some of the other applications, very briefly.

2 Applying CSP/FDR to a Tactical Internet

2.1 The First Attempt

We had access to descriptions of a tactical internet technology; this was being developed by a leading networking firm and it was tendered as one component of the UK Army's next generation communications system. The technology was required to create and to maintain a radio network across highly mobile nodes, in another words it was a MANET.

The descriptions covered various aspects of the network, but of greatest interest, to us, were the networking protocols. Our first attack on the problem was to code the networking protocols in CSP_M. This was challenging because the descriptions available were somewhat incomplete. Nevertheless, we were able to produce CSP_M models of the system. The models were, in essence, a direct translation, of the networking protocols. Of course, data types were limited to small finite types and continuous parameters were discretised to small finite ranges, but, apart from that, not much abstraction was applied. As such, we created *high fidelity* models of the system.

Unfortunately, the state space of these models was too large to perform much meaningful analysis. While FDR is equiped with a number of state reduction operators [4], these were not able to improve matters significantly. The models of the networking protocols were simply too complex.

2.2 Results with Abstract Models

While the high fidelity models were being developed, another thread of work had been studying routers exchanging link state data. Link state routing is one of the major types of routing protocol [1]. In a link state protocol, each router maintains a database of the state of all links in the network.

To study link state routing CSP_M was used to describe link state routers, in a network of variable topology, trying to agree on the network topology. This work was the basis for the CSP_M specifications of self-stabilisation described below. These models were very abstract; they did not attempt to model any specific protocol. Instead, they simply had routers exchanging time-stamped information about the state of specific links.

The tactical internet used link state routing protocols which were closely related to IP standards. Some state changes, in its networking protocols, were decided according to nodes' views of the network topology. It was possible to upgrade our CSP_M models of routers with a representation of some of these node states. In effect, this yielded an abstract model of aspects of the tactical internet's functionality. Analysing these models proved to be interesting, it showed that the network could partition itself, furthermore some partitions could fail to be detected - some of these behaviours were not at all obvious. We discussed this first phase of the work with the designers of the tactical internet. These behaviours were known to them (in fact an extra layer of protocol coped with some of them), but they were impressed by an automated capability to find protocol flaws and they were very positive about this work. Thus the primary benefit of this phase was to provide us with an encouraging proof-of-concept.

We had found that the high fidelity models were largely intractable, and that starting with very abstract models (of routers exchanging link state data) and adding selected features was promising. This led us to model the mobile network, as a whole, at an abstract level. The essential idea behind the abstraction was to represent a sub-protocol, which caused a set of nodes to change their states with a single CSP event. Such events are synchronised across nodes and they change the nodes' states according to the effects of the sub-protocol. In essence, nodes are given the capability to update states of other nodes directly. This modelling was the basis for the powerful and generic techniques discussed in Section 4, below.

The more abstract modelling started with the simplest possible representation of the substates of a node, with single events controlling transitions between substates. This model was refined incrementally, the pattern was that an abstract CSP network model would be found to have some CSP divergence (some cycle of internal events). This corresponded to the potential for part of the network to continually re-configure itself. This is a type of self-stabilisation failure, and it could be removed by refining the model by adding more system features.

This type of modelling soon developed into an interesting analysis. The models were gradually refined to contain a representation of most of the networks behaviour. In these quite rich models FDR found complex configurations of network topology and node state, which in conjunction could cause the network to perpetually re-configure itself.

Figure 1, attempts to give a flavour of the type of behaviours found by FDR in our models of the tactical internet. Each node decides its state change on the basis of:

- its own state
- the state of its neighbours
- its knowledge of the network topology

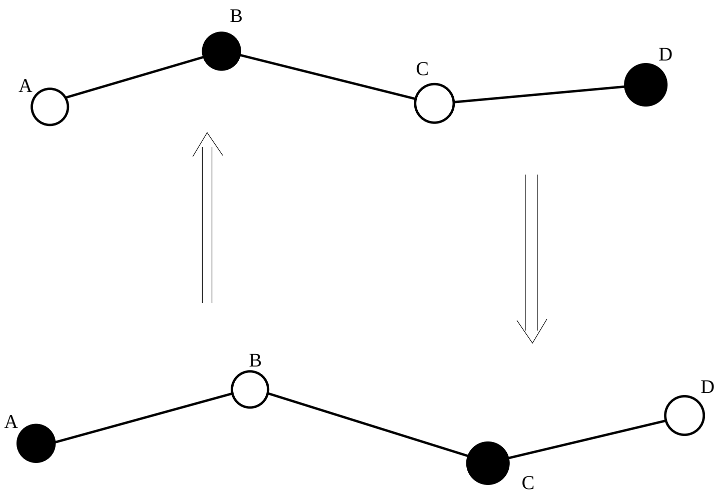

Fig. 1. Simple example of cyclic re-configuration in a MANET

In effect FDR explores all possible combinations of these factors, which can lead either to cycles of re-configuration, or to the network configuring itself incorrectly (see section 5.3). In our simple example, the nodes in state *White* decide, on the basis of the above factors to change to state *Black*; and *vice-versa*. The result is a global configuration that is symmetric to the previous state. Each node, will then make corresponding state changes that return the network to the original configuration, and so on.

The result of this modelling was to identify the need for a number of randomising elements in the network specification. Certain state changes in the protocols were guarded by a randomised delay. These delays were found at a few, very select, places in the network description. Initially, we simply ignored them, we could not guess their purpose. However, as our abstract models developed, we found that these randomising elements were necessary to preclude possibilities of perpetual re-configuration, such as alluded to by our fig 1. Returning to our example, if either pairs of nodes A and C, or B and D are forced to wait a random amount of time, before changing their states, then the simultaneity that engenders the cycle is broken.

We subsequently discovered that using a randomised delay is a standard technique for avoiding these potential self-stabilisation failures. However, *finding* the very select places to put these randomised delays is difficult. Current methods are based on a combination of expertise and extensive simulation.

2.3 Using Parallel FDR

The main aim of this paper is to present the abstract modelling techniques, but in this subsection we note how we were later able to make progress with the high fidelity models. During the project we had contacts with Jeremy Martin, then at the Oxford University super-computing centre. He had been experimenting with parallelising FDR and he was subsequently very successful in completing a full parallel implementation of FDR [8].

On the high fidelity models this more powerful verification engine yielded, as expected, a more detailed analysis of self-configuration problems. In fact, most of these were complex transient states that the network could find itself in, due to nodes having stale information about the network. Unfortunately, it is not possible to discuss these behaviours without exposing sensitive information about the tactical internet.

3 Principles of the Methodology

The methodology for modelling mobile networks, that was inspired by the tactical internet study, is simple in concept and powerful in applicability; it also has the benefit of not suffering, very much, from the state explosion. The driving idea is to capture the states of the system's components simply and then to map interactions between components onto their states. The methodology depends on two features of CSP_M, *viz.*:

1. CSP_M's programming support for renaming processes through a relation - this is used to program the way nodes interact.
2. CSP's, hence CSP_M's, model of shared event communication - this is used to implement interactions between nodes in our models.

Renamings are a very powerful programming construct in CSP_M. If P is a process and R is a relation on sets of events, then renaming P through R, written $P[[R]]$, maps P to its relational image under R. So, if a is one of the events performed by P, then $P[[R]]$ will offer the external choice of all the events in $\{x \mid aRx\}$, instead of a. Furthermore, if a set, E, of events are renamed, by R, to the single event e, then the occurrence of e in $P[[R]]$ corresponds to the non-deterministic choice of the E-events in P. In general, events can be renamed in many ways and in this work we primarily use one-to-many renamings.

CSP uses a handshake model of communication. If processes P and Q communicate on an event a, then both P and Q perform a single a together. Furthermore, communication is multiway, so many processes can synchronise on events in the same way as two processes.
The methodology has three parts:

1. **The local view.** The base of the model are CSP_M processes that capture the state transitions of the units of the subject; crucially this includes processes for links as well as nodes. These basic processes only capture *local* states and state transitions.
2. **Promoting local to global.** The CSP_M renamings that map interactions onto the states of the basic processes. In other words the local states of the node and link processes are promoted to the ways they can affect each other. In effect renamings promote a *local view* to a *system view*.
3. **Specifying properties.** In FDR properties are also specified in CSP_M, but CSP has no built in notion of time or temporality. Nevertheless, we require our subjects to be eventually 'correct' (cf. Perlman's notion of self-stabilisation). In fact, we can code the requisite properties of eventual 'correctness', quite elegantly, in CSP.

An important part of our approach is to condense as much protocol as possible into a single shared event. It is also important to make the 'local view' processes as simple as possible. Clearly, the extent to which this is possible will vary from problem to problem, and it will also vary according to the skill of the CSP_M programmer. To illustrate the three parts of the methodology on CBRP we first need a brief description of it.

4 A Short Description of CBRP

The Cluster-Based Routing Protocol [5] is a networking and routing protocol designed for use in MANETs. It uses distributed algorithms to organise nodes into clusters. Each cluster has one, and only one, head, and a number of member nodes. Thus the states of a node are either a cluster *Head*, or a cluster *Member*, but, in certain circumstances, a node can have a third *Undecided* state, before deciding to become a *Head* or a *Member*. The clusters are identified by the ID of their *Head*, and a node is deemed to be a *Member* of a cluster if and only if it has a bidirectional link to the *Head*. So a node can be a *Member* of several clusters simultaneously.

Nodes detect the presence of other nodes and organise themselves into clusters by using regular broadcasts known as 'HELLO' messages. These are periodically broadcast from each node. As a node receives these messages, it builds up a table of the nodes which it is able to hear, and the state of these neighbouring nodes (i.e. *Head*, *Member*, or *Undecided*). The HELLO messages are made up of the state of the broadcasting node together with the neighbour table. By observing whether or not its own ID appears in a neighbour table, a node can determine whether it has a unidirectional or a bidirectional link the sender of the HELLO message.

Cluster formation is also centred around the HELLO messages. Initially, all nodes are in the *Undecided* state, and all nodes commence their HELLO message broadcasts. A timer is also started at each node. If a node receives a broadcast from a *Head* before the time-out, then it becomes a *Member*. If it times out without hearing such a broadcast, then it can automatically go from the *Undecided* to *Head*. If several non-*Head* nodes are in bidirectional contact with each other, then the node with the lowest ID becomes the *Head*. To assist nodes in moving from *Undecided* to *Member*, a *Head* node will, in addition to its periodic broadcasts, send out a triggered HELLO message whenever it receives a broadcast from an *Undecided* node. Once all nodes are in either *Head* or *Member* states, cluster formation is complete.

CBRP is intended for operation in a dynamic environment and the network of clusters must be maintained as connectivity changes; thus nodes states are also liable to change, as radio connectivity changes. Cluster maintainence follows much the same pattern as cluster formation. If a *Member* node loses its last bidirectional connection to a *Head* node, then it will revert to the *Undecided* state and follow the initial procedure. Also, if a *Head* gains a bidirectional link with another *Head*, for longer than a predetermined length of time, then the node with the lower ID remains a *Head*, and the other becomes one of its *Member* nodes. Furthermore, if several non-*Head* nodes come into contact with each other, then the node with the least ID in the peer group becomes a *Head*.

Let us note that network formation and maintenence in CBRP is rather simpler than in the tactical internet. For instance, in CBRP state changes in a node are decided by its own state and that of its peer group; but in the tactical internet, a node's knowledge of the network topology is an additional factor in nodes deciding state changes. The motivation for creating the clusters is to support efficient routing; but we are not going to discuss that, instead we will concentrate on how our methodology enables us to model CBRP's dynamic cluster formation and maintenence.

5 Modelling CBRP

We are now ready to describe how CBRP can be modelled. Abstraction is an important part of our methodology; modelling CBRP starts with the following abstractions:

- We assume that links are always bi-directional (this can easily be rectified, but our presentation will make this assumption).
- We do not store and communicate the neighbour table, but it is possible to access neighbour information implicitly, through the link processes.
- We make a node always receptive to any communication that will change its state (instead of modelling the time triggered and event triggered broadcast of 'HELLO' messages).

5.1 State Transitions of the Basic Processes

The Basic Node. A node will have a number of states. In the case of CBRP, these states are *Head*, *Member* and *Undecided*; so we have the CSP_M type declaration:

$datatype\ NodeState = Head \mid Member \mid Undecided$

The basic node process is parameterised by the current *NodeState*. It performs two types of event: the *stay* event identifies the *current* state the node is in, and it does not change the node's state; with a *move* event the node changes its state, and the event carries its *current* state and its *new* state (the *diff* function is simply set difference).

```
channel stay : NodeState
channel move : NodeState.NodeState

BASICNODE(current) =
    stay.current → BASICNODE(current)
      □
    move.current?new : diff(NodeState, {current}) → BASICNODE(new)
```

The Basic Link. We capture a lot of interesting dynamic behaviour by having a simple two state process for a link, these states are *Up* and *Down*. Transition between these two states is controlled by the *make* and *break* events and *linkstate* events report whether the link is *Up* or *Down*.

```
channel make, break
channel linkstate : LinkState

BASICLINK =
    let
    DOWN =
      make              → UP
        □
      linkstate.Down    → DOWN
    UP =
      break             → DOWN
        □
      linkstate.Up      → UP
  within
    DOWN
```

BASICLINK has *UP* and *DOWN* as its local states, with *DOWN* being the initial state.

BASICNODE and *BASICLINK* are the fundamental units of the system's model; but to build the model these processes need to interact and the node and link identifiers need to be added. The interactions are programmed by renaming the basic processes according to the ways they can affect each others state, and, as such, these renamings also carry the node and link identifiers. The interactions are then implemented by making the renamed CSP_M processes communicate.

5.2 Renaming and Connecting Processes

Renaming and synchronisation are used to create the model of the network in a number of ways and the principal techniques are summarised here. We assume that nodes are named A, B, etc. and that this defines the type *NodeId*. Elements of this type are ordered alphabetically. We can then identify a link by the pair of nodes it connects, viz.: $\{A, B\}$,$\{B, D\}$, etc.

Causing State Changes. A local event can be renamed to a cause of a state change, in another node. The local event will typically capture the local state of the node. In the CBRP example the *stay.x* event, in the *BASICNODE*, is renamed to cause all state changes that are a consequence of state x. For instance, suppose we want to program the way a node, A, can affect other nodes when it is a *Head*. For this we use the channel:

channel announce : *NodeState.NodeId*

to rename *stay.Head* to *announce.Head.A*, in the code for node A. But to complete mapping the effects of a node being a *Head* we need to make local state changes, in other nodes, receptive to all events of type *announce.Head*.

Allowing Local States to be Changed. A local state change is marked by the appropriate event and this event must be renamed to allow other nodes to activate the state change. So if a node, B, is *Undecided*, then it will become a

Member when its link with a *Head* node becomes *Up*. To implement this the *move.Undecided.Member*, in node B, has to be renamed to:

$\{announce.Head.x \mid x \in diff(NodeId, \{B\})\}$

To iterate, applying this renaming means that in node C the external choice of both the above events will be offered. So in the *Undecided* state C will synchronise with either *announce.Head.A* or *announce.Head.B* to become *Member*.

Modelling Synchronised Broadcast. Network connectivity is another factor in deciding how nodes affect each other and this determines the various ways the *BASICLINK* process (or its analogue) is renamed.

To continue our example of the interactions between *Head* and *Undecided* nodes, to allow A to broadcast that it is a *Head* all links in the set:

$\{\{A, n\} \mid n \in diff(NodeId, \{A\})\}$

will have their *linkstate.Up* event renamed to *announce.Head.A*. This in renaming, in conjunction with the two former renamings, allows a *Head* node to change all *Undecided* nodes, of lower id, into *Member*. In effect we have implemented a synchronised broadcast by parameterising the *announce.Head* event with only the sender.

Modelling Pointwise Interactions. *Undecided* nodes also broadcast their state to solicit a respose from a neighbouring *Head*; the response is, in effect, a point-to-point communication. To model the reponse from a *Head*, we need a new channel:

$channelpt2pt : Sender.Receiver.NodeState$

where both *Sender* and *Receiver* are equal to *NodeId*. Taking the instance where A is *Head* and B is *Undecided*, renaming each of:

- *stay.Head* in the process for node A,
- *move.Undecided.Member* in the process for node A,
- *linkstate.Up* in the process for link $\{A, B\}$

to *pt2pt.A.B.Head*, will implement this part of the protocol. In general pointwise interactions are modelled by parameterising the 'global view' event with both sender and receiver.

Note that the sub-protocol we have described is condensed into the single *pt2pt* event. Finally, note that the *Undecided* broadcast does not cause any node's state to change and an *Undecided* node is always receptive to a change to *Member* by a *pt2pt* event. This means we can omit the *Undecided* broadcast and condense this sub-protocol into the single *pt2pt* event.

Capturing Effects of Losing Connectivity. The renamings we have discussed above have depended on links being *Up*. We can capture the consequences of loosing connectivity with renamings as well, here are two brief examples.

When a *Member*, say C, loses its last link to a *Head* it becomes *Undecided*. This can be implemented by declaring:

channel losehead : *NodeId*

and renaming C's *move.Member.Undecided* event to a *losehead*.C event. Also all *linkstate.Down* events in C's links must be renamed to the same *losehead* event. Thus if node C is a *Member* and it has lost all its links to its *Head*'s, then it will synchronise on a *losehead* event, with its link processes, to become *Undecided*.

The *linkstate.Down* event can also be used to simulate a timeout. If an *Undecided* node does not get a link to a *Head*, then it times out and becomes a *Head*. Maintaining our nomenclature, if the *move.Undecided.Head* event is also renamed to *losehead*.C, then this will synchronise with the *losehead*.C events in C's links to make C a *Head*.

Using Multiway Synchronisation. Multiway synchronisation can be used to condense a complex interaction across nodes into an atomic global state transitions. In the case of CBRP, multiway synchronisation can be used to model, with a single transition, when a group of mutually audible *Undecided*'s elect the one with the lowest ID as a new *Head*, while the rest become its *Member*.

One might imagine that this would require accumulating state at each node to make this decision. However, the flexibility of synchronisation in CSP allow this to be achieved in a single multi-way communication; but this multiway communication involves not only the nodes in the connected component of the network in question, but those isolated from it as well.

When a link is *Down*, the link process prevents its nodes from being elected, but it permits any other node to be made the *Head*. When the link is *Up*, the nodes, at the end of the link:

– prevent any election, when they are *Head*;
– allow themselves or nodes of lower id to be elected, when they are *Undecided*;
– block the election of any higher ID, when they are *Undecided*.

Now, when the intersection of all of these sets is calculated, by the semantics of parallel composition, the result is the singleton contain the lowest ID of the peer group. Thus each *Undecided* node is left with a single possibility : if its ID is the one in the singleton, then it becomes *Head*; but if its ID is higher than the chosen one, then it becomes a *Member*.

Exposing State for Property Checking. Finally, the connection between this step of the method and the last step is to expose the state of nodes at the global level. This enables the specifications, described in section 5.3, to be coded and model-checked against. For this we need the channel:

$channel\ report : NodeId.NodeState$

In each node the *stay* event is renamed to the *report* event. Thus in node A, $stay.x$ is renamed to $report.A.x$.

With our model of CBRP (or whatever self-configuring system we happen to be studying) complete, we need the last step of our methodology: a formalisation of the properties it must satisfy.

5.3 Specifying and Checking Properties

FDR is a *refinement checker*, it checks whether one CSP_M process, the specification, or *SPEC* is *refined* by another, the implementation, or *IMP*. Refinement is with respect to one of the (three most common) denotational semantics of CSP, viz. traces, failures and failures-divergence. And to say that *IMP* refines *SPEC* simply means that the denotational value of *IMP* is a subset of the denotational value of *SPEC*.

We want to check that our system will always terminate its self-configuration, resulting in a 'correct' state. In effect, the CSP_M specification must capture the following:

- The self-configuration terminates.
- The resulting distributed state of the system is 'correct'.

But to prevent FDR stopping a check when the system reaches a transient bad state, the specification must also:

- Permit the system to be in an incorrect state prior to the self-configuration terminating.

FDR establishes the refinement relationship by comparing the operational forms of *SPEC* and *IMP* - which are labelled transition systems. Operational states are either *unstable* - in which case they can perform hidden events, or they are *stable* - in which case they only do visible events. To verify a failures-divergence assertion:

1. FDR checks unstable states of *IMP* for cycles of hidden events, namely *divergences* that are disallowed by *SPEC*'s unstable states.
2. FDR checks that the stable states of *IMP* can refuse no more than *SPEC*'s stable states.

Now consider the following CSP process:

$SPEC = right \rightarrow SPEC \ \Box \ (STOP \sqcap wrong \rightarrow SPEC)$

This process:

1. is divergence free, or simpy there is no cycle of hidden events.
2. it only refuses *wrong* events.
3. it will permit *wrong* events to occur.

In a node only the *report* events are visible, all self-configuration events are hidden, thus a divergence, in *IMP*, corresponds to a failure to stabilise. Furthermore, *right* and *wrong* events are mutually exclusive subsets of the *report* events. *SPEC* allows *right* and *wrong* events in unstable states and only *right* events in stable states. Checking that the refusals of stable states are no more than all *wrong* events corresponds to requiring that every single *right* event is allowed in the stable states. A stable state is one in which no further self-configuration is possible, so self-configuration must terminate in states where all *report*s are everything *right*. So to establish the failures-divergences assertion:

$SPEC \sqsubseteq_{FD} IMP$

IMP must always eventually terminate its self-configuration in the *right* state. In fact, using this style of specification requires additional finessing by making the *SPEC* insensitive to the topology changing events. But with that done a wide range of properties can be verified of a model, for CBRP these include:

– no node is left in the *Undecided* state,
– every connected component has a *Head*.

6 Conclusions

We have dicussed work that we did to model and analyse a tactical internet using CSP_M and FDR. And we have described the methodology for modelling ad-hoc networks and self-configuring systems, that were inspired by the tactical internet work. The methodology allows tractable models of complex dynamic behaviour to be created quickly and easily. It is our belief that the power of this methodology depends on a combination of the simplicity and flexibility of the fundamental processes (section 5.1) and the powerful programming constructs supplied by CSP_M, primarily the renamings (section 5.2). This is confirmed by our experience with these techniques on other problems.

For instance, to model the use of mobile IP in partitionable networks, the basic concepts clearly apply, with appropriate changes. So the *BASICNODE* process has to be tailored for each of the participants, viz: the mobile agent, the home router, the remote host and the message; but it is very similar to what we have described here. The partitionable connectivity can be captured by having a *BASICLINK*-type process for each partition. The interactions between these fundamental process is mapped onto them by the appropriate renaming relations. Coding a property, such as a message must always reach the mobile agent, requires an eventual settling specification (section 5.3). Model-checking then shows how a partition can make a message bounce between the home router and a formerly occupied remote host.

We also found this approach very useful when modelling the self-configuring key hierarchy of [6] (this could be regarded as an ad-hoc key hierarchy). There a group of nodes maintain a key hierarchy without a central server; the hierarchy re-configures itself as the group partitions and as partitions heal. Once again processes very similar to the *BASICNODE* and *BASICLINK* are very useful. They model the state transitions of groups of nodes and group keys. In the protocol, the groups of nodes and the group key processes must interact as the connectivity varies and the appropriate renaming relations can capture these interactions. The result is an elegant and efficient model that captures the full dynamic behaviour of the protocol.

With regards to related work, in this broad area we are only aware of Khurshid and Jackson's work on [9], but that is not closely related to this work. As far as we are aware there is not much work applying model-checking to MANETs. We feel this work is also interesting because model-checking applications of this sort are uncommon.

7 Acknowledgements

The authors would like to thank Will Simmonds, Nick Moffat and Sadie Creese for helpful comments on drafts of this paper.

References

1. Perlman, R. *Interconnections: Bridges, Switches and Routers.* Addison-Wesley, 1999.
2. `http://www.cs.uiowa.edu/ftp/selfstab/bibliography/`
3. `http://www.ietf.org/html.charters/manet-charter.html`
4. Roscoe, A.W. *The Theory and Practice of Concurrency.* Prentice-Hall, 1998.
5. `http://www.comp.nus.edu.sg/~tayyc/cbrp/`
6. Rodeh, O., K.Birman, D.Dolev. Optimized Group Rekey for Group Communication Systems. Network and Distributed System Security, 2000.
7. Creese, S. *Data Independent Induction : CSP Model-checking of Arbitrary Sized Networks.* DPhil thesis, University of Oxford, Compting Laboratory, 2001.
8. Martin, J., M.H.Goldsmith, *et al.* Parallel FDR. Formal (Systems) Europe Ltd. Technical Report, forthcoming.
9. Khurshid, S., D. Jackson. Exploring the Design of an Intentional Naming Scheme with an Automatic Constraint Analyzer. Proc. 15th IEEE International Conference on Automated Software Engineering, Grenoble, France, September 2000.

A A Short Introduction to CSP

This appendix gives a brief summary of the CSP operators used in the text. In practise modelling is done in CSP_M, which is a machine readable version of CSP, embedded in a full functional programming language.

$a \rightarrow P$ This process performs event a, then proceeds according to process P.

$P \;\square\; Q$ The external choice of P and Q, the environment chooses from the initials of P or of Q.
$P \sqcap Q$ The non-determinstic (or internal) choice between P and Q. The environment is offered either the initials of P or of Q.