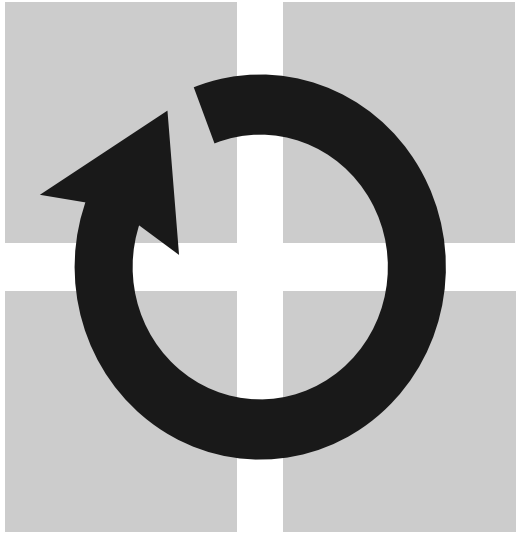
MAFTIA

Malicious- and Accidental-Fault Tolerance for Internet Applications

Systemmodellierung, Middleware-Architektur, Verifikation



Hans Reiser

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Universität Erlangen-Nürnberg

Überblick

- Was ist MAFTIA?
- System-Modellierung

  Fehler, Synchronität, Topologie, Interaktionen, Gruppen
- Middleware-Architektur von MAFTIA
- Verifikation und Bewertung

Was ist MAFTIA?

- MAFTIA: Malicious- and Accidental-Fault Tolerance for Internet Applications
- EU-Forschungsprojekt, Laufzeit 1.1.2000-31.12.2002, Projektmittel: 5,1 Mio. Euro
- Projektpartner:
  - University of Newcastle upon Tyne (UK)
  - Faculdade de Ciencias, Universidade de Lisboa (P)
  - DERA, Malvern (UK)
  - Universität des Saarlandes (D)
  - CNRS-LAAS (F)
  - IBM Zurich Research Lab (CH)
- **`http://www.newcastle.research.ec.org/maftia/`**

Systemmodell: Fehler-Modellierung

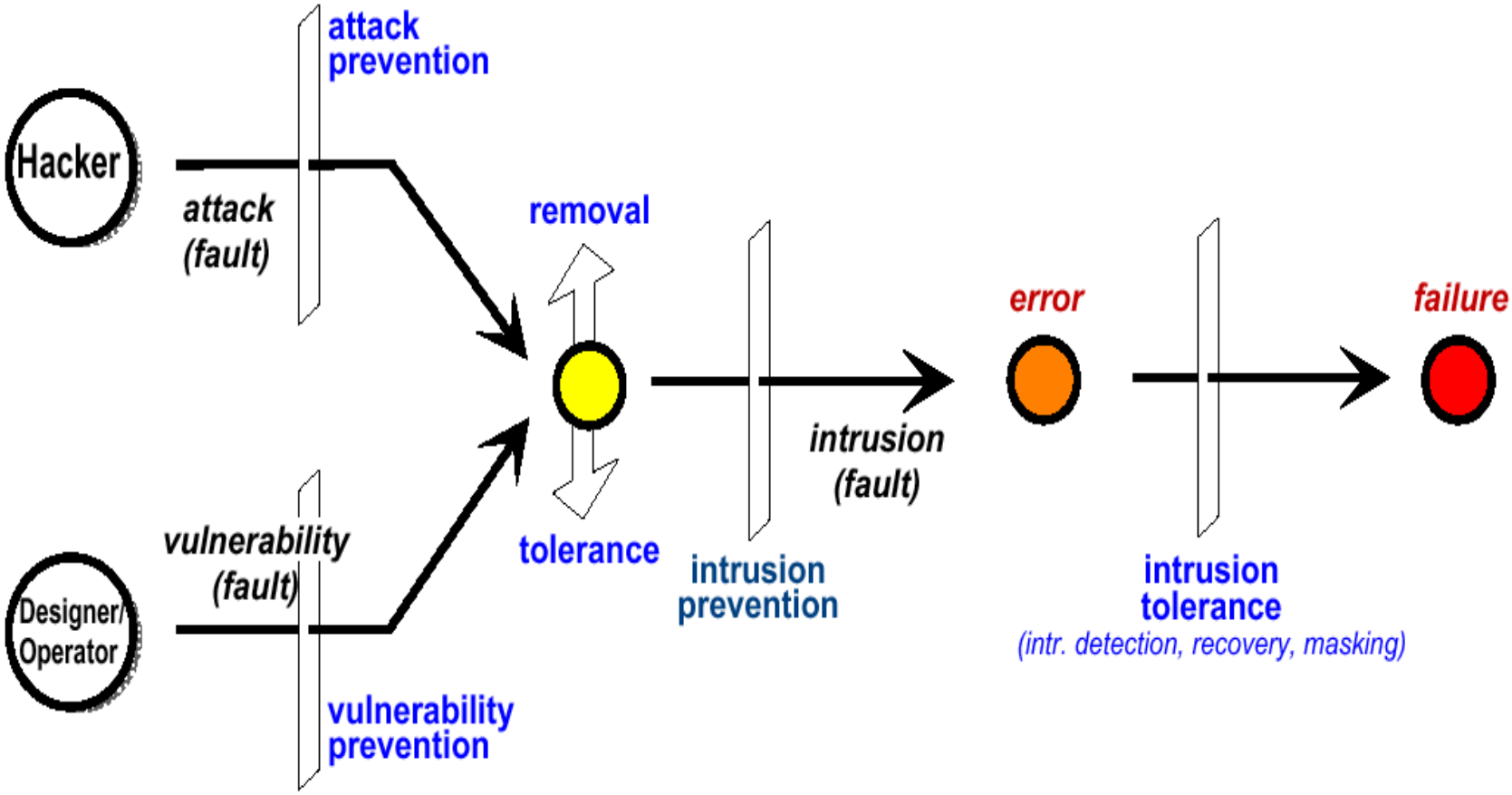
- Begriffsdefinitionen
  - “fault”: Ursache eines “errors”
  - “error”: Teil d. Systemzustands, der zu “failure” führen kann
  - “failure”: System erfüllt seine Aufgaben nicht richtig

Systemmodell: Fehler-Modellierung

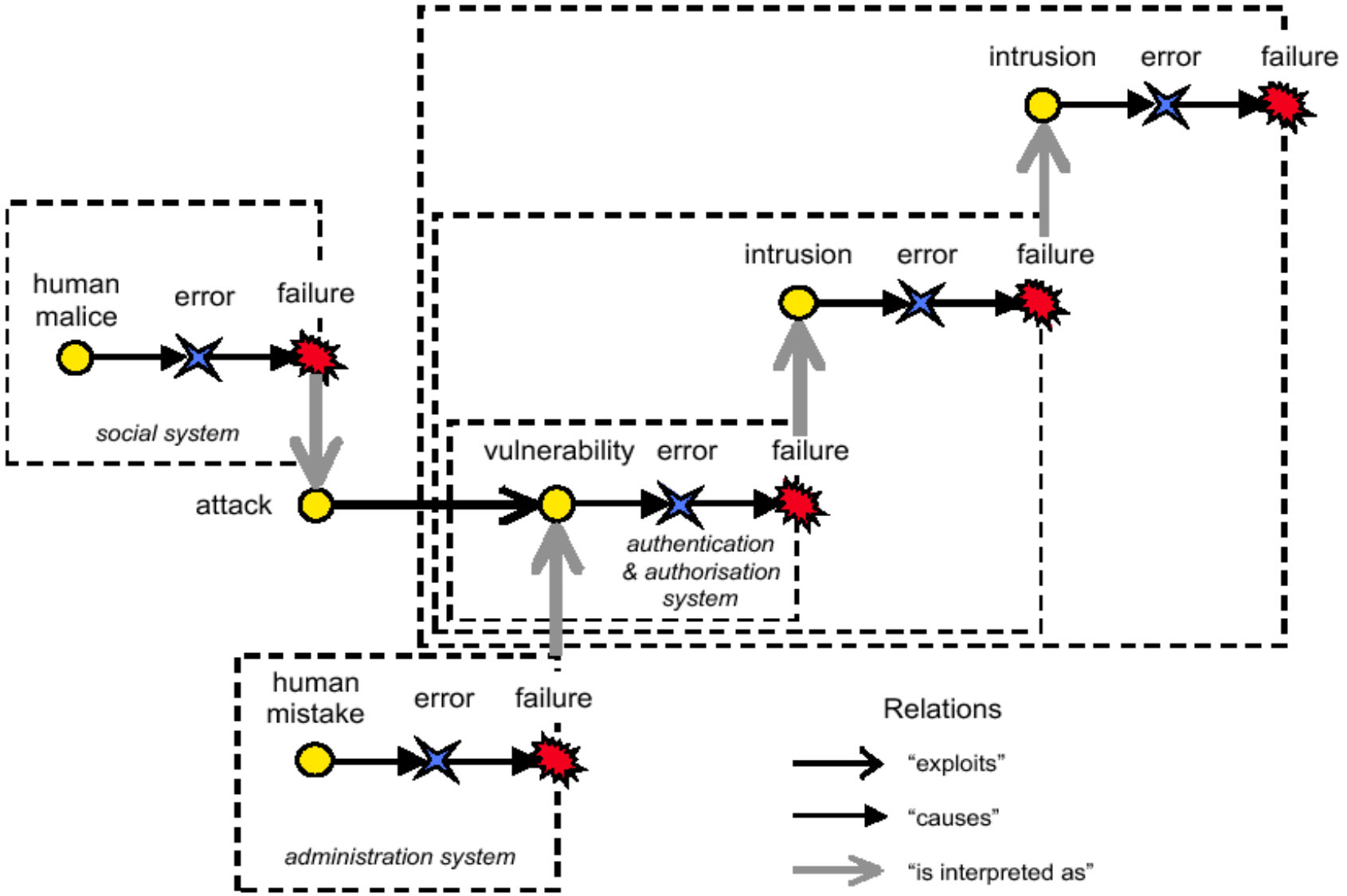
- Begriffsdefinitionen
  - “fault”: Ursache eines “errors”
  - “error”: Teil d. Systemzustands, der zu “failure” führen kann
  - “failure”: System erfüllt seine Aufgaben nicht richtig
- Beispiel:
  - “fault”: Sicherheitsloch in meinem System
  - “error”: Angreifer dringt in mein System ein
  - “failure”: Angreifer legt mein System lahm

Systemmodell: Fehler-Modellierung

- Mehrstufiges Fehlermodell von MAFTIA



Systemmodell: Fehler-Modellierung



Systemmodell: Arten von Fehlern

- Kontrollierte Fehler:
  - Qualitative und quantitative Schranken für Fehler von Komponenten (“gutmütig”, “bösartig”)
  - Realistische Modellierung von zufälligen Fehlern möglich
  - Verhalten von gezielten Angriffen schwierig zu erfassen

Systemmodell: Arten von Fehlern

- Kontrollierte Fehler:
  - Qualitative und quantitative Schranken für Fehler von Komponenten (“gutmütig”, “bösartig”)
  - Realistische Modellierung von zufälligen Fehlern möglich
  - Verhalten von gezielten Angriffen schwierig zu erfassen
- Willkürliche Fehler:
  - Fehler ohne Einschränkungen, z.B. jederzeit beliebige Nachricht an beliebigen Knoten senden
  - Wenn überhaupt, in der Praxis nur mit großem Aufwand realisierbar

Systemmodell: Arten von Fehlern

- Hybrides Fehlermodell:
  - Kombination:
    Unterschiedliche Annahmen für unterschiedliche Teile des Systems
  - Gut bei modular strukturierten Systemen zu realisieren
- Möglichst genaues Fehlermodell wünschenswert!!!

  (aber schwierig...)

Systemmodell: Angreifer und Fehler

- Behandlung/Charakterisierung von möglichen Angriffen
  - *“Kerckhoff’s assumption”:* Angreifer kennt vollständige Spezifikation vs. “*security by obscurity*” (z.B. GSM, DVD)

Systemmodell: Angreifer und Fehler

- Behandlung/Charakterisierung von möglichen Angriffen
  - *“Kerckhoff’s assumption”:* Angreifer kennt vollständige Spezifikation vs. “*security by obscurity*” (z.B. GSM, DVD)
  - Replikation als Basistechnik (bis zu *t* von *n* Knoten unter vollständiger Kontrolle des Angreifers)

Systemmodell: Angreifer und Fehler

- Behandlung/Charakterisierung von möglichen Angriffen
  - *“Kerckhoff’s assumption”:* Angreifer kennt vollständige Spezifikation vs. “*security by obscurity*” (z.B. GSM, DVD)
  - Replikation als Basistechnik (bis zu *t* von *n* Knoten unter vollständiger Kontrolle des Angreifers)
  - Betrachtung des Netzwerks als Teil das Angreifers

Systemmodell: Angreifer und Fehler

- Behandlung/Charakterisierung von möglichen Angriffen
  - *"Kerckhoff's assumption":* Angreifer kennt vollständige Spezifikation vs. "*security by obscurity*" (z.B. GSM, DVD)
  - Replikation als Basistechnik (bis zu *t* von *n* Knoten unter vollständiger Kontrolle des Angreifers)
  - Betrachtung des Netzwerks als Teil das Angreifers
  - Recovery & kryptographische Verfahren: Wissen des Angreifers kann nicht rückgängig gemacht werden! "*proactive security*"

Systemmodell: Synchronität

- Asynchron vs. Synchron:
  - Verarbeitungszeiten
  - Kommunikationsverzögerung
  - Drift der lokalen Uhren
  - Abweichung der einzelnen Uhren voneinander

Systemmodell: Synchronität

- Asynchron vs. Synchron:
  - Verarbeitungszeiten
  - Kommunikationsverzögerung
  - Drift der lokalen Uhren
  - Abweichung der einzelnen Uhren voneinander
- Aussagen über Zeit notwendig:

  Quality-of-Service-Anforderungen benötigen Zeit

  Lösbarkeit bestimmter Probleme erfordert Zeit (-> 29.4.)
- Aussagen über Zeit problematisch:

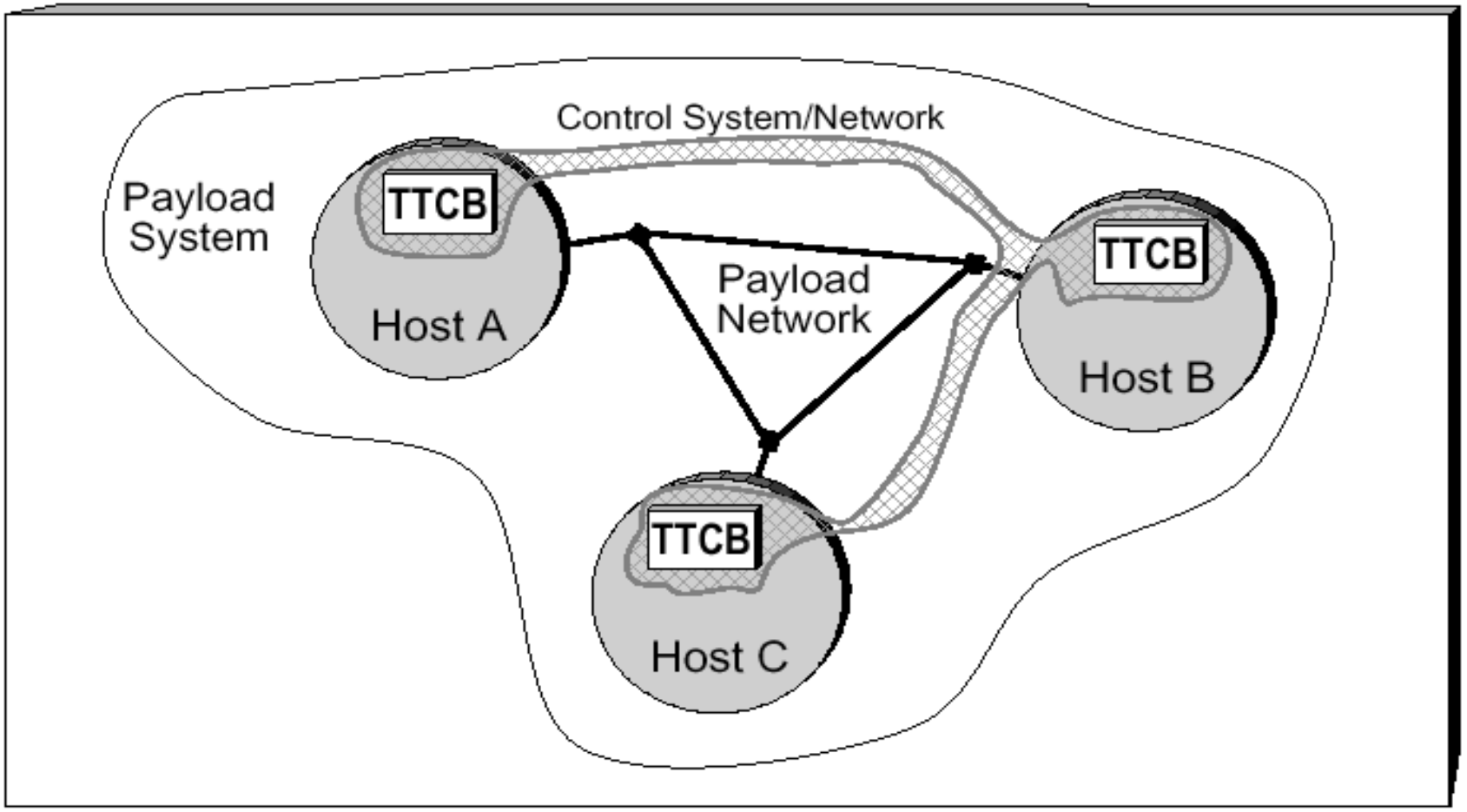
  Abhängig von Infrastruktur, erhöhte Angreifbarkeit

Systemmodell: Synchronität

- beliebtes Modell: Partielle Synchronität
  - Zeitbezogene Spezifikationen erlaubt
  - Verletzung der zeitlichen Spezifikationen vorgesehen
  - Mechanismen zur Erkennung & Behandlung von zeitlichen Fehlern

Systemmodell: Synchronität

- MAFTIA: Trusted Timely Computing Base (TTCB)



Systemmodell: Synchronität

- MAFTIA: Trusted Timely Computing Base (TTCB)
  - *trusted timely execution, trusted absolute timestamping, trusted duration measurement, trusted timing failure detection*
  - zeitliche Fehler durch einen bekannten Wert beschränkt
  - Realisierbar durch
    - Internet-basiert: RTP, IP QoS
    - Externe Dienste: Echtzeit-LAN, ISDN-Direktverbindungen, LEOs

Systemmodell: Synchronität

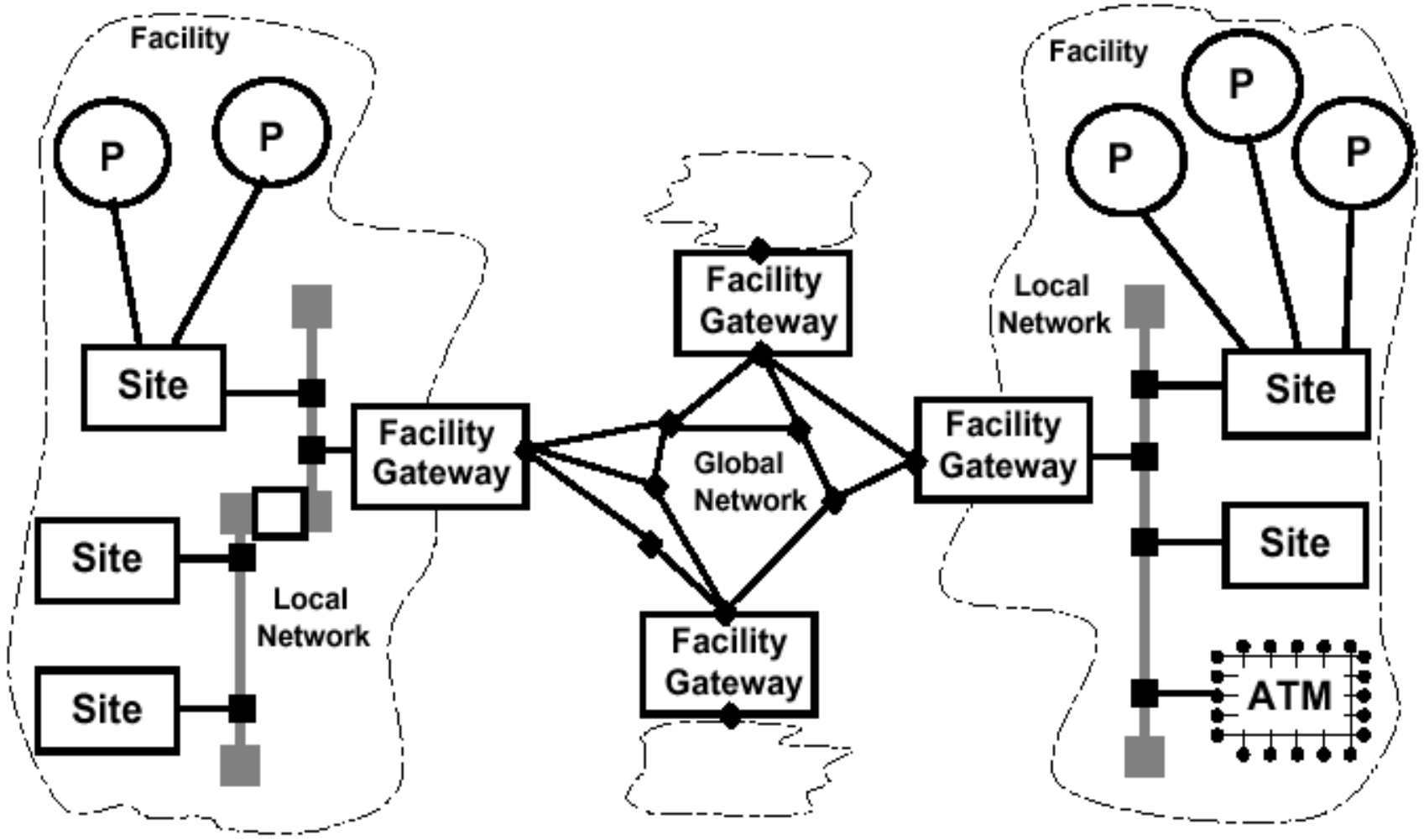
- MAFTIA: Alternative: Zeitloser Ansatz
  - Verwendung von randomisierten (probabilistischen) Algorithmen
  - Aussagen zur Terminierung nur probabilistisch möglich
  - ... siehe Vortrag am 3. Juni

Systemmodell: Topologische Struktur

- Vorteile von Topologiebewusstsein
  - Aufbau einer für die Anwendung vorteilhaften Topologie
  - Entwurf von Algorithmen und Protokoll so, dass diese die Topologie günstig ausnützen
- Gruppierung (Clustering):
  - “divide-and-conquer”-Strategie: Skalierbarkeit!
  - Rekursiv auf mehreren Ebenen

Systemmodell: Topologische Struktur

- Zweistufige Struktur: WAN von LANS



Systemmodell: Interaktionsstrukturen

- Client/Server
  - RPC oder Nachrichtenbasiert (Gruppenkommunikation)
- Multipeer
  - Spontaner, symmetrischer Nachrichtenaustausch zwischen einer Menge von Knoten
- “Dissemination”
  - Publisher-Subscriber-System (Subscriber push oder pull) (weniger Sender, viele Empfänger)
- Transaktionen
  - “ACID” (atomicity, consistency, isolation, durability)

Systemmodell: Gruppen

- Offene vs. geschlossene Gruppen

  Wer kann/darf mit einer Gruppe kommunizieren?

- Statische vs. dynamische Gruppen

  Dynamische Änderung der Mitgliedschaft zur Laufzeit?

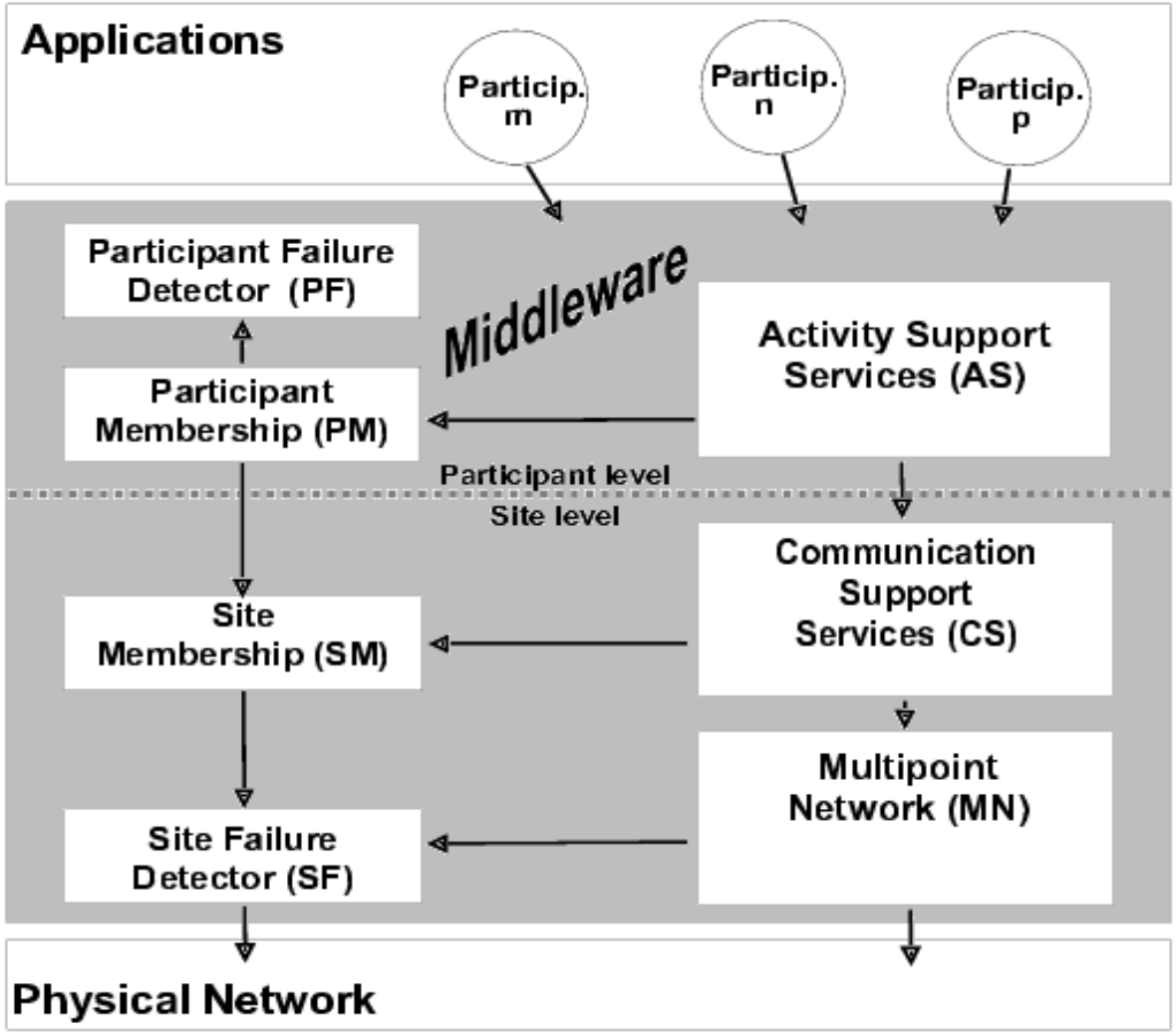
- Gruppenmitglieder aufnehmen und entfernen

  geeignete Strategien/Protokoll erforderlich!

MAFTIA-Architektur

- Übliche generische Dienste in verteilten Systemen:
  - Namensdienst
  - Kommunikationsdienste (Netzwerk)
  - RMI
  - Zeit-Synchronisation
  - Verteilte Dateisysteme
  - Brockerage Service (ORB)
  - Registrierung, Authentifizierung, Authorisierung
  - Administrative Dienste

MAFTIA-Architektur



Verifikation

- Formale Verifikation ist Voraussetzung für Vertrauen
- Präzises formales Systemmodell erforderlich
  - fail-stop und bösartiges Fehlverhalten
  - Angreifer mit beschränkter/unbeschränkter Rechenleistung
  - statische und dynamische Angreifer
  - formale Modelle für Synchronität
  - formale Modelle für Topologie und Interaktionsarten

Verifikation

- Ziel der Spezifikation
  - Was soll mein System tun (-> Integrität, Verfügbarkeit)
  - Was dürfen andere nicht von meinem System erfahren (-> Vertraulichkeit)

Verifikation

- In MAFTIA:
  - Verwendung von CSP (bekannt z.B. aus KS1?)
  - Geeignete Modellierung von Angreifern aufwendig, aber anscheinend machbar.
  - Aktueller Stand:
    - Modell für sychrones Netz mit Angreifer
    - Modell für “synchronous contract signing”

Zusammenfassung

- Entwicklung fehlertoleranter Software erfordert zunächste geeignete Modellierung.

  * Fehlerarten * Synchronität * Netzwerk-Topologie *

  * Interaktionsstrukturen * Kommunikationsgruppen *

- Geeignete Middleware-Dienste sind wünschenswert.

- Formale Verifkation ist sinnvoll und möglich, aber aufwendig.